NASP–9114–03
NAS Enroute Stage A
Contract FA65WA–1395

4

NAS OPERATIONAL SUPPORT SYSTEM

SUBPROGRAM DESIGN DOCUMENT

IBM 9020 Data Processing System
Basic Assembler Language Program (BALASM)

Model A3d2.1

15 Aug
~~24 May~~ 1974

This document provides detailed design information to aid the programmer in the maintenance of the Basic Assembler Program.

These change pages update this document to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 tape release.

NAS Programming
IBM Federal Systems Division
NAFEC, Atlantic City, New Jersey

CHANGE HISTORY

This (03 level) publication of the BALASM SDD is current with the NOSS composite tape labeled N3D21x.

Level	Date	Change Proposal Number	Comment
01	15 August 1973		Original Publication
02	24 May 1974		A3d2.1 Update
03	15 August 1974		A3d2.1 Update

x = latest tape level

PREFACE

This document, prepared by the International Business Machines Corporation, is submitted to the Federal Aviation Administration in accordance with the requirements of Contract FA65WA-1395.

These change pages update the NAS Operational Support System (NOSS) Subprogram Design Document (SDD) for the IBM 9020 Data Processing System Basic Assembler Language Program (BALASM), dated 24 May 1974, to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

ILLUSTRATIONS

ILLUSTRATIONS (Continued)

CHARTS

CHARTS (Continued)

CHARTS (Continued)

CHARTS (Continued)

CHARTS (Continued)

CHARTS (Continued)

CHARTS (Continued)

TABLES

ABBREVIATIONS

ABKSET	This routine developes the reference table.
ABS8	Evaluate Absolute Expression Subroutine
ANALY	Condense and Sort Routine
ASMCON	This subroutine sets up a constant generated for alignment and places it in the print area.
ASMLOC	This subroutine sets up the current location counter value and places it in the printer area.
ASUBRT	This routine evaluates A and S constants.
BAL	Basic Assembly Language; Master Control and Communications Route
BALASM	Basic Assembler Language Program
BEGIN	New Statement Routine
BLBRKUP	Break Up Expression Routine
BLCONMOD	Constant Modifier. This mainline routine controls BLCONMOD processing.
BLCONVAL	Evaluate Constant Routine
BLEXVAL	Evaluate Expression Routine
BLIOGET2	Get a Statement During BLPAS2 Routine
BLIST	Listing Generator Routine
BLPAS1	This routine is the main body of the assembler first pass.
BLPAS2	This routine is the main body of the assembler second pass.
BLPUNC2	This routine punches TXT, ESD, RLD, DBG, and END cards, and links to the PAKTAP subroutine.
BLOPLKUP	Operation Code Lookup Routine
BLSLKUP	Symbol Table Search Routine

BSUBRT	This routine evaluates F, H, E and D constants.
CCI	This routine counts the number of EBCDIC characters specified in the value list of a C-type constant.
CCW	Channel Command Word. In BLPAS1 this subroutine saves an aligned doubleword space for the channel command word. In BLPAS2 this subroutine validates and assembles a channel command word.
CERR	This routine generates a diagnostic code.
CMPLST	Compool Documentation Program
CNOP	Conditional No Operation. In BLPAS1 this subroutine validates the operands of a Conditional No Operation pseudo-operation. In BLPAS2 this subroutine assembles one or more NOP instructions to align the location counter.
COM	This subroutine initiates or resumes the common section.
COMMEN	Print a Comment Record Subroutine
COVX	Convert Fullword to EBCDIC Subroutine
CSECT	Control Section. This subroutine initiates or resumes the control section names in the CSECT statement.
CTFULL	This subroutine processes CSECT-EXTRN table overflow.
DBLWRD	This subroutine aligns current location counter to a doubleword boundary.
DCDS	This subroutine evaluates the operand of a DC or DS statement.
DCL	This subroutine issues diagnostic 35 when a DCL source statement is encountered.
DCODS	This subroutine reserves storage space for its operand and sets up object code for the constant value.
DEBUGG	Debugging System Program

DIAG	Stack A Diagnostic Subroutine
DIAGX	This subroutine prints any diagnostic messages for the input statement.
DOUB	Align To Doubleword Subroutine
DROP	This subroutine validates the register operands of a DROP statement.
DS	This subroutine prints DS statements.
DS1	This subroutine handles DS statements when there is no listing.
DSECT	This subroutine initiates or resumes the control section names in the DSECT statement.
DSUBRT	This subroutine calls EXVAL to evaluate an expression and checks for errors.
DUMP	This subroutine validates a storage and/or register dump request.
DUMPT	This subroutine validates a tape dump request.
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJECT	This subroutine causes the listing to skip to a new page.
END	In BLPAS1 this subroutine recognizes the end of the source program and exits to the interlude. In BLPAS2 this subroutine validates the 'begin' location, puts out entries, and exits, ending BLPAS2.
ENTER	The first pass subroutine uses control card information to select the source for statements.
ENTRY	This subroutine evaluates and enters symbols in the ENTRY table.
EOF	End-of-File
EQMAMI	This subroutine enters the statement name and location in the symbol table.
EQMXMN	This subroutine assembles the location specified by the statement START, EQU, MAX, or MIN.

ESD	External Symbol Dictionary
EXBC/EXBCR	These subroutines assemble extended mnemonic branch instructions.
EXTE	EXTRN Routine
EXTRN	This subroutine validates the operands of an EXTRN statement.
EXVR	Evaluate Expression Subroutine
FROMD	Convert to Floating Point Routine
FROMF	Convert to Fixed Point Routine
HALF	This subroutine aligns location counter to halfword boundary.
HEX	Hexadecimal
HEX2CM	This subroutine converts hexadecimal numbers into EBCDIC characters.
HFWRD	This subroutine aligns current location counter to a halfword boundary.
HHI	This routine counts the number of hexadecimal characters specified in the value list of an X-type constant.
IGNORE	This subroutine handles statements that are ignored by BLPAS2.
ILLOP	This subroutine is called for an invalid operation code.
INCSTA	Convert to EBCDIC Subroutine.
INIT1	PASS1 Initialization Subroutine
INIT3	PASS2 Initialization Subroutine
INIT4	PASS3 Initialization Subroutine
INT8	Evaluate 8-Bit Immediate Operand Subroutine
INROUT	Evaluate An Integer Subroutine
INTGER	This subroutine sets up an expression for evaluation by the DSUBRT subroutine and checks for errors.

I/O	Input/Output
IOGET1	Get A Statement During PASS1 Routine
IOPUT1	Put Out A Statement During PASS1 Routine
ISEQ	This subroutine initiates or suppresses sequence number checking.
JOVIAL	Jules' Own Version of International Algorithmic Language
LHOLD	This routine develops the length, according to type, for a constant in which length is implied.
LIB	This subroutine moves characters from operand of LIB statement into a card image and punches a LIB card.
LIBEDT	Library Edit Program
LIST	This subroutine temporarily suppresses or resumes listing.
LITYP	Evaluate Register Operand No. R2 Subroutine
LLNGTH	This routine determines the byte and bit length of a constant using the explicit length given in the input statement.
LOADER	Loader Program
LOCOVF	Process Location Counter Overflow Subroutine
LOC1B	Dump-Trace Routine
LOC1F	Three-Character Operation Code Routine
LOC2	Five-Character Operation Code Routine
LOC3	Two-Character Operation Code Routine
LOC4A	CSECT-DSECT-TEQU Routine
LOC10	Locate Symbols Routine
LTORG	This subroutine puts out all literals in the literal table as DCL statements and aligns the location counter and prints the LTORG statement.
LTRTN	This subroutine processes literals.

MASTXX	Reference-Table-Code Translate Routine
MAST0	Reference Table Overflow Routine
MAST3	Double-Overflow Routine
MBRKUP	This subroutine calls BRKUP and evaluates expressions.
MEVAL	This subroutine calls EXVAL and processes diagnostics.
MHOLD	This routine assigns a multiplicity of 1 when no explicit value is specified.
MISPL	This subroutine handles statements that are disallowed by the system or that should not have reached BLPAS2.
MLC	Merged Library/Compool
MMULTY	This routine determines multiplicity when it is specified by the input statement.
MPUTD	This subroutine puts in a diagnostic record.
MPUTS	This subroutine puts in a statement record.
NAS	National Airspace System
NLIST	This subroutine temporarily suppresses or resumes listing.
NOAN	End-Of-Job Subroutine
NOSS	NAS Operational Support System
NUCD	This subroutine is used to fill out and punch a card during DC, DCL, and alignment statement processing.
ORG	This subroutine evaluates the ORG statement operand and prints the ORG statement.
OVERFL	This routine checks for a reference table overflow.
PAKTAP	This subroutine writes a card image or packed TXT record on .AUXIL.
PAKTXT	This subroutine packs TXT cards into a buffer.

PASS1	Initialization Subroutine
PASS2	New Statement Routine
PR	Print Routine
PREAD	Compool Read Subroutine
PRINT	This mainline routine identifies the input statement and calls subroutines to prepare a print line.
PRNT	This subroutine validates the operands of a PRINT statement.
PRTESD	This subroutine prints ESD line.
PRTIT	This subroutine prints a line in the assembly listing.
PSEG	This subroutine validates the operands of PSEG cards.
PSEUDO	Pseudo-Operation Routine
PUN	This subroutine handles the punching of TXT cards.
PUCD	This subroutine punches the information stored in the card area.
PUTESD	This subroutine puts the ESD cards.
QUAL	This subroutine validates a new qualifier and puts it in the communications region.
QRLD	This routine requires an entry for a relocatable address constant.
RDMRG	Read and Merge Source Statements Routine
READ	This subroutine selects source on first pass and reads a statement.
REGST	Evaluate Register Operand Subroutine
RLD	Relocation List Dictionary
ROUT	This subroutine sets up a print line for a constant requiring two or more lines and moves it to the print area.

RPEM	This subroutine suppresses or resumes the printing of possible error messages.
SA	Evaluate Storage Address Operand Subroutine
SA4Z	Evaluate Storage Address With 4-Bit Length = 0 Subroutine
SCANX	This subroutine evaluates an element of an expression.
SCH2	Search Routine
SDD	Subprogram Design Document
SE	Storage Element
SEQ	This subroutine checks the sequence number of a statement when necessary and sets a flag in the listing if an error is found.
SGET	This subroutine finds the location of an entry in the symbol table that matches a specified symbol.
SHFT	Evaluate Shift Operand Subroutine
SI8Z	Evaluate Storage Address With 8-Bit Length = 0 Subroutine
SKIP	This subroutine steps to the next word in the output listing and sets up for a new element.
SLA	Evaluate Storage Address With 8-Bit Length Subroutine
SLA4Q	Evaluate Storage Address With 4-Bit Length Subroutine
SPACE	This subroutine causes the listing to space a specified number of lines.
SPEM	This subroutine suppresses or resumes the printing of possible error messages.
SSCALE	This routine evaluates scaling and initiates value list handling.
SSEQ	This subroutine initiates or suppresses sequence number checking.

SRTRT	Prepare for SORT Routine
START	This subroutine evaluates and saves the starting location to be assigned for the object program.
STA2PR	This subroutine sets up a statement for printing and moves it to the print area.
STFULL	This subroutine processes symbol table overflow.
STNTRY	This subroutine creates a symbol table entry.
STORE	This subroutine moves a character to the output list and increases counters to prepare for the next character.
SXXX	Locate Acceptable Statements Routine
SYMVAL	This subroutine validates a statement symbol.
SYSEDT	System Edit Program
TEQU	This subroutine changes the location attribute of the symbol named in the TEQU statement and prints the TEQU statement.
THOLD	This routine sets up for a C type constant when no explicit type is given.
TITLE	This subroutine validates the punch ID and causes the title page of the listing to be printed.
TRACE	This subroutine validates and assembles the trace label.
TTYPE	This routine determines the type of constant specified by the input statement.
USBAS	This subroutine develops base and displacement for an S-type address constant.
USBASE	Compute Base and Displacement Subroutine
USING	This subroutine validates the value and registers specified in the USING statement.
UTILITY	Utility NOSS Monitor Program
VVI	This routine checks the number of constants in the value list of an A, S, F, H, E, or D type constant.

XA	Evaluate Indexable Storage Address Subroutine
XREF	Compool Reference Matrix Program; Punch XRF Cards Routine
XRFBEG	This subroutine processes each symbol reference and builds the XRF symbol card image.
XRFINIT	This subroutine initializes for the XREF function.
XRFPCH	This subroutine punches cards for the XREF function.
XRFTRL	This subroutine provides termination of the XREF function.
ZZI	This routine finds the total number of characters in the value list of P and Z type constants.

1.0 INTRODUCTION

1.1 PURPOSE AND SCOPE

This manual provides detailed information on the internal logic of the assembler program. This program converts source programs in Basic Assembly Language (BAL) into object language.

This publication is intended for technical personnel who are responsible for analyzing program operations, diagnosing them, or adapting them to special uses.

1.2 RELATIONSHIP TO OTHER DOCUMENTS

This document assumes that the reader is familiar with the IBM 9020 Data Processing System Basic Assembly Language User's Manual.

1.3 REFERENCE DOCUMENTS

Effective use of this manual requires an understanding of IBM 9020 System operation, of the IBM 9020 Utility Programming System Basic Assembly Language, and of the functions of the 9020 Utility Programming System. This information is available in the following publications:

a. IBM System Reference Library, IBM 9020 System Principles of Operation, Form A22-6852, Order Number ZA22-6852-latest revision, Contract FA64WA-5223, IBM Corporation, Federal Systems Division, 18100 Frederick Pike, Gaithersburg, Maryland 20760.

b. IBM 9020 Data Processing System: User's Manual, Basic Assembly Language (BALASM), NASP-9214-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

c. NAS Operational Support System User's Manual, Debugging System (DEBUGG), NASP-9216-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

d. NAS Operational Support System User's Manual, Library Edit Program (LIBEDT), NASP-9220-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

e. NAS Operational Support System Subprogram Design Document, Compool Documentation Program (CMPLST), NASP-9105-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

f. NAS Operational Support System User's Manual, Utility NOSS Monitor (UTILITY), NASP-9229-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

g. NAS Operational Support System Subprogram Design Document, Compool Reference Matrix (XREF), NASP-9112-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

h. NAS Operational Support System User's Manual, System Edit Program (SYSEDT), NASP-9222-latest revision, Contract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

i. NAS Operational Support System User's Manual, Loader Program (LOADER), NASP-9219-latest revision, ontract FA65WA-1395, IBM Corporation, Federal Systems Division, NAFEC, Atlantic City, N. J.

This manual is a comprehensive guide to the logical structure and functions of the 9020 Data Processing System Assembler Program. It is designed to be used with the assembly listing and, consequently, does not discuss program structure at the machine instruction level. Symbols used in this manual correlate with those in the listing.

1.4 DOCUMENT SUMMARY

This manual is organized into several major sections. Excluding the introductory section, the remaining sections cover the following portions of the assembler.

Section 2.0, Pass 1 Description, presents BLPAS1 and other routines that comprise the first pass of the assembler.

Section 3.0, Pass 2 Description, presents BLPAS2 and the routines used in pass 2 of the assembler.

Section 4.0, Symbolic Analyzer Description, presents a description of the symbolic analyzer, BLANALYZ.

Section 5.0, Storage and Timing, lists the storage requirements and timing estimates.

Section 6.0, Data Specifications, presents the following data specifications.

a. Subsection 6.1 contains the formats of entries in each major table used or built by the assembler. These tables include the symbol table, the CSECT-EXTRN table, the ENTRY table, the literal table, the RLD table, the USING table, the operation code table, the symbolic analyzer tables, and the segment table.

b. Subsection 6.2 contains descriptions of data and record formats.

c. Subsection 6.3 describes the input and output formats for certain key routines: BLEXVAL, BLCONMOD, BLCONVAL, FROMD, and FROMF.

d. Subsection 6.4 gives the code and test of all diagnostic messages issued by the assembler.

1.5 SUBPROGRAM DESCRIPTION

a. Purpose

The IBM 9020 Assembler converts a source program in BAL~~X~~ into object language suitable for loading and execution. The source program may be one of the following:

1. A compiled JOVIAL program.

2. A program deck written in assembly language.

3. Compool reserves for one Compool segment.

At the user's option, the assembler produces one or more of the following:

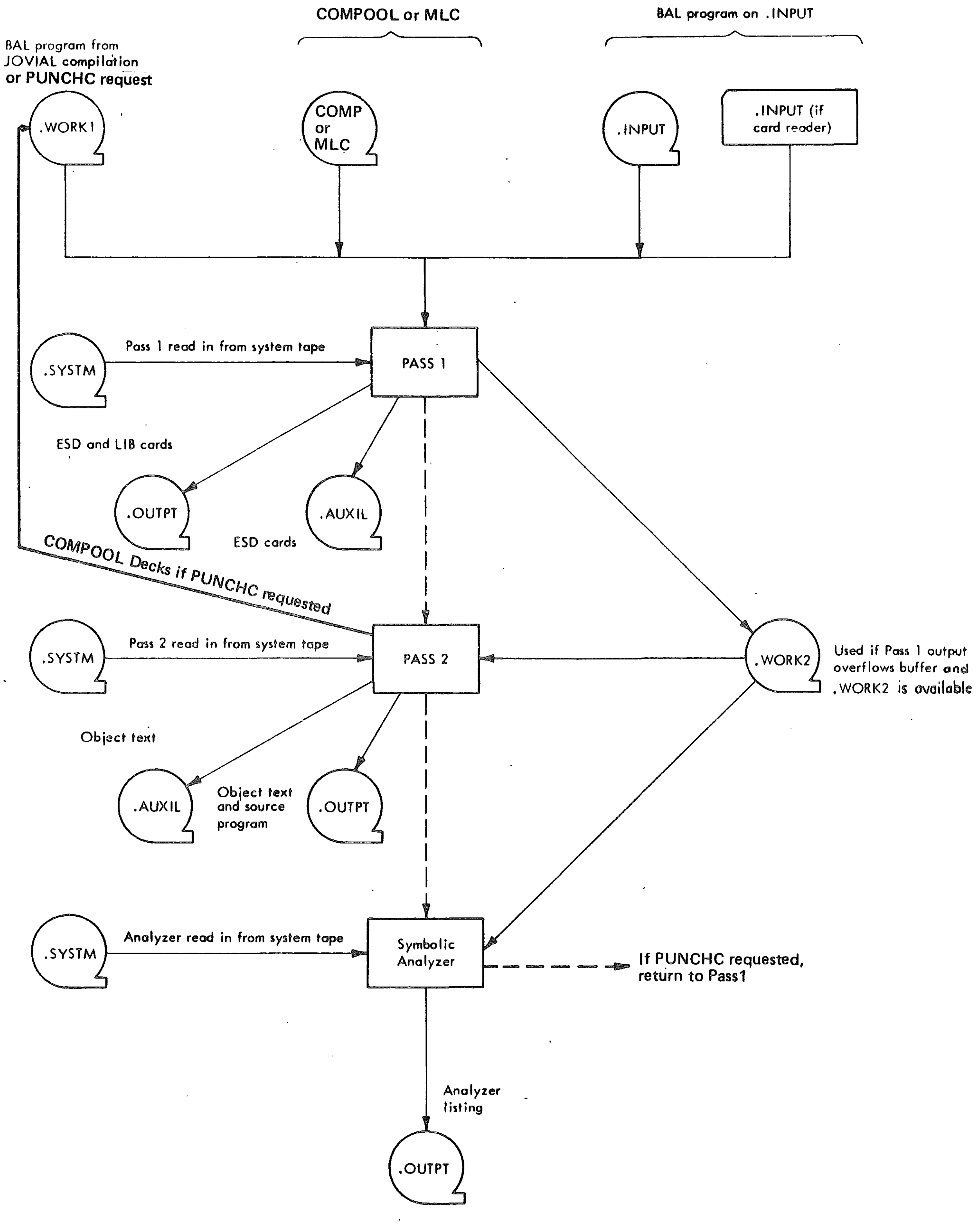
1. Object text for immediate loading and execution.

2. A listing of the assembled program, with or without a cross-reference symbol listing (produced by the symbolic analyzer).

3. An object-language punched card deck.

b. Operating Characteristics

The assembler is executed in two passes (or three if the symbolic analyzer is requested, see Figure 1-1).



COMPOOL or MLC
BAL program on .INPUT
BAL program from JOVIAL compilation or PUNCHC request
.WORK1
COMP or MLC
.INPUT
.INPUT (if card reader)
.SYSTM
Pass 1 read in from system tape
PASS 1
ESD and LIB cards
.OUTPT
.AUXIL
ESD cards
COMPOOL Decks if PUNCHC requested
.SYSTM
Pass 2 read in from system tape
PASS 2
.WORK2
Used if Pass 1 output overflows buffer and .WORK2 is available
Object text
.AUXIL
Object text and source program
.OUTPT
.SYSTM
Analyzer read in from system tape
Symbolic Analyzer
If PUNCHC requested, return to Pass1
Analyzer listing
.OUTPT

Note: .OUTPT is generally processed off-line to produce printed listings and punched card decks. If the card-reader/punch and printer are on-line, taking the place of the .OUTPT tape, printing and punching are performed directly.

FIGURE 1-1. IBM 9020 ASSEMBLER INPUT/OUTPUT FLOW

Pass 1 examines each source statement in turn to assign it a tentative location and to build control section, symbol, and literal tables. Possible and serious errors are noted for issuance of diagnostic messages during pass 2. Pass 1 concludes by assembling the control sections in order of their definition, updating the symbol table, and putting out the External Symbol Dictionary (ESD). If a compiled JOVIAL program being assembled contains procedures designated for addition to the program library, LIB cards are produced by pass 1 for submission of the procedure(s) to the library edit program.

Pass 2 converts each source statement into object language, using the operation code table and the tables created by pass 1 as well as the USING table it develops to perform base-displacement address calculations. After the last source statement has been processed, pass 2 puts out the Relocation List Dictionary (RLD) and a count of serious and possible errors (even if no listing was requested). Assembly is complete at this point, unless a cross-reference symbol dictionary was requested; if so, the assembler proceeds to pass 3.

Pass 3 (symbolic analyzer) uses the tables built in pass 1 to develop a cross-reference listing of symbol definitions and references, as well as a list of undefined symbols.

c. Assembler Program Organization

The assembler is organized physically on the system tape (.SYSTM) into three records named BAL, BALPT2, and BALPT3. The first record, BAL, contains the PASS1 routine plus nine separately assembled routines. These nine routines each perform a unique function (e.g., symbol-table lookup, expression evaluation, etc.). The BALPT2 record, which contains the PASS 2 routine plus three separately assembled routines, is brought into storage at the end of the first pass, overlaying those parts of the BAL record that are no longer needed. If the symbolic analyzer (BALPT3) is requested, it is brought into storage after the second pass is completed, overlaying part of BALPT2.

The three assembler records and the routines they contain are described briefly below. Refer to Figure 1-2 for an overview of the assembler logic flow.

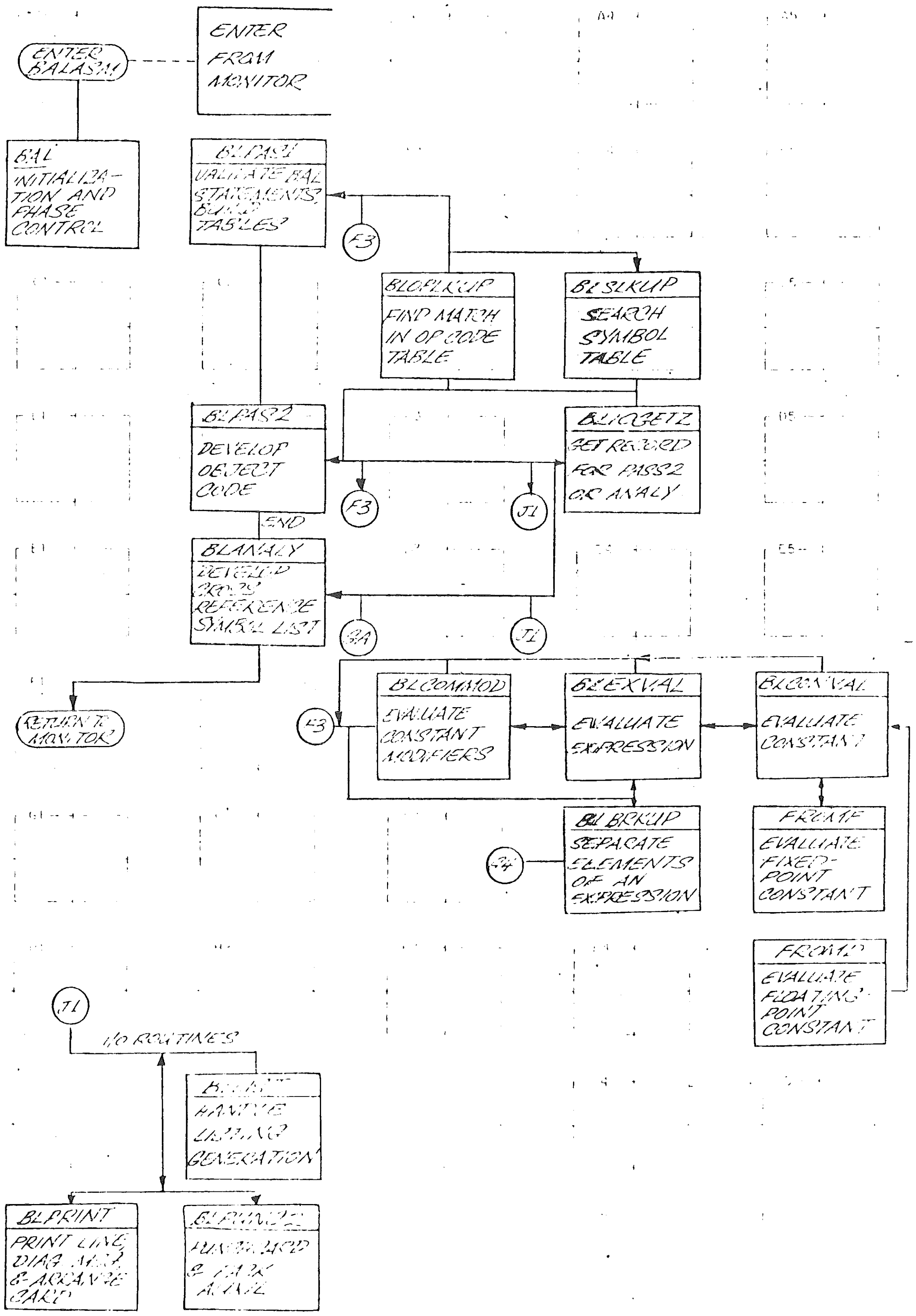

ENTER BALASM
ENTER FROM MONITOR
BAL
INITIALIZATION AND PHASE CONTROL
BLPAS1
VALIDATE BAL STATEMENTS; BUILD TABLES
F3
BLOPLKUP
FIND MATCH IN OP CODE TABLE
BLSLKUP
SEARCH SYMBOL TABLE
BLPAS2
DEVELOP OBJECT CODE
BLIOGETZ
GET RECORD FOR PASS2 OR ANALY
F3
J1
END
BLANALY
DEVELOP CROSS REFERENCE SYMBOL LIST
3A
J1
RETURN TO MONITOR
F3
BLCOMMOD
EVALUATE CONSTANT MODIFIERS
BLEXVAL
EVALUATE EXPRESSION
BLCONVAL
EVALUATE CONSTANT
BLBRKUP
SEPARATE ELEMENTS OF AN EXPRESSION
3A
FROMF
EVALUATE FIXED-POINT CONSTANT
FROMD
EVALUATE FLOATING-POINT CONSTANT
J1
I/O ROUTINES
HANDLE LISTING GENERATION
BLPRINT
PRINT LINE, DIAG MSG & ARRANGE CARD

FIGURE 1-2. ASSEMBLER LOGIC FLOW

Record	Program or Routine
BAL	BLPAS1 - This is the mainline routine of pass 1, which examines each source card and assigns it a tentative location within its control section in the program being assembled. A symbol table, a control-section/EXTRN table, an ENTRY table and a literal table are built. After all source cards have been examined, the control-section/EXTRN table is updated, making each control section start just after the end of the preceding one. Symbol locations in the symbol table are similarly increased by the amount their control-section start address has been increased. Finally, ESD cards are prepared for use in relocating the assembled program when it is loaded. This routine also contains all the code concerned with reading input from SYSIN or the compool, and code to write the intermediate work file on WORK2.
	BAL - This routine contains the overall coordination for the assembler. It handles initialization as well as sequencing the various passes.
	BLPUNC2 - This routine punches an object deck when requested. If the job is to be executed immediately, TXT cards, in condensed form, are also placed on .AUXIL.
	BLOPLKUP - This routine uses a binary search to find the statement operation code in the operation code table in MASTER.
	BLLIST - This routine prints the listing for all portions of the assembler. It includes page overflow and numbering logic and handles spacing.
	BLSLKUP - This routine uses a 'hashing' algorithm to select an entry point in the symbol table, then searches until the symbol or an unused entry location is found.
	BLBRKUP - This routine separates elements of an expression and performs a syntactical check on these elements.

Record	Program or Routine
	BLEXVAL - This routine examines an expression for errors, then evaluates it.
	BLCONMOD - This routine checks modifiers (multiplicity, type, length, scaling) of a constant or literal.
	ZXMASTER - This is the communications region of the whole assembler.
BALPT2	BLPAS2 - This is the mainline routine of pass 2. It is brought into storage after BLPAS1 is completed, overlaying BLPAS1. This routine converts each source statement into object language, computes base-displacement addresses, and issues diagnostic messages for errors found in the source program.
	BLCONVAL - This routine evaluates a constant and checks it for consistency. A subroutine builds the RLD table for relocatable address constants. It contains FROMF which converts a constant in EBCDIC to fixed-point doubleword form, and FROMD which converts a constant in EBCDIC to floating-point long form.
	IOGET2 - This routine gets one record from the intermediate buffer (or .WORK2) for the second pass or for the symbolic analyzer.
	PRINT - This routine arranges a statement and/or object code for printing. It then arranges the object code into TXT cards. Finally, it uses the diagnostic code(s), if any, to get a message to be printed on the listing following the statement.
BALPT3	BLANALYZ - If a cross-reference symbol listing is requested, this routine is brought into storage after BLPAS2 is completed, overlaying BLPAS2. It produces the listing of symbol definitions and references and a listing of undefined symbols.

2.0 PASS 1 DESCRIPTION

a. Function

Pass 1 of the IBM 9020 Assembler builds tables for use by pass 2 (and pass 3 if requested) and determines the length of the object program in the course of examining all source program statements. Each source statement is read, put into canonical form, assigned a tentative location within its Control Section (CSECT), and then put into the intermediate buffer. When that buffer is filled, succeeding statements are put on WORK2 unless it is not available.

As pass 1 proceeds, it builds up a control-section/ EXTRN (CSECT-EXTRN) table, a symbol table, an ENTRY table, and a table of literals. After all source statements have been read, pass 1 updating the CSECT-EXTRN table by assigning the location following the end of the first control section as the starting address of the second control section, and so on. The locations assigned to the symbols defined in each control section are similarly increased by the starting address of that control section. Finally, ESD cards, and a listing of them, are prepared from the CSECT-EXTRN table and from the ENTRY table, concluding pass 1.

General logic flow and detailed flowcharts of the programs that comprise PASS 1 have been included at the end of this section.

b. Organization

The initialization code and resident coordinator are contained in Routine BAL. This is contained in System Record BAL along with the Pass 1 Mainline Code (BPAS1), the communications area (ZXMASTER), and seven common routines. The latter remain resident throughout the assembly. The BPAS1 code is overlaid by later passes.

2.1 BAL ROUTINE

The BAL (master control and communications) routine contains the communications region used by all assembler routines, the tables used by other routines (see Subsection 6.1), and four subroutines. Of the four subroutines, three are for initialization, and one is the end-of-job subroutine. The names and functions of these subroutines are as follows:

1. INIT1 - This subroutine initializes for PASS1 and processes program options. It goes on completion to BLPAS1.

2. INIT3 - This subroutine initializes for BLPAS2 and brings it in.

3. INIT4 - This subroutine initializes for the BLANALYZ program and brings it in if the symbolic analyzer was requested. If not, this subroutine exits immediately to the NOAN subroutine.

4. NOAN - This subroutine processes PUNCHC requests or terminates the assembler operation.

2.1.1 PASS1 Initialization Subroutine (INIT1)

FUNCTION: This subroutine (Chart AA1) initializes for PASS1.

ENTRY: This subroutine is entered at INIT1 from the Monitor.

OPERATION: This subroutine loads the communications region base register and loads the system Input/Output (I/O) routine addresses in that area. It next uses machine memory size to look up table sizes and define the tables to be built up by the assembly operation. If .WORK2 is not available, the table sizes are made larger because the entire program must be contained in core storage.

The $BAL control card information is processed and End-of-File (EOF) returns for tapes are set. Initialization is performed for punching and listing.

EXIT: This subroutine exits to BLPAS1.

ERRORS: Errors found by this subroutine produce diagnostic messages without stopping execution.

2.1.2 PASS2 Initialization Subroutine (INIT3)

FUNCTION: This subroutine (Chart AA2) initializes for BLPAS2, brings in that program, and turns over control to it.

ENTRY: This subroutine is entered at INIT3 from the end of BLPAS1.

EXIT: This subroutine exits to the beginning of BLPAS2.

2.1.3 PASS3 Initialization Subroutine (INIT4)

FUNCTION: This subroutine (Chart AA2) initializes for the BLANALYZ routine and brings it in if the symbolic analyzer was requested. If not, this subroutine exits immediately to the end-of-job subroutine (NOAN).

ENTRY: This subroutine is entered at INIT4 from the end of BLPAS2.

EXITS: This subroutine exits to the NOAN subroutine for end-of-job or to the BLANALYZ program.

2.1.4 End-of-Job Subroutine (NOAN)

FUNCTION: This subroutine (Chart AA3) checks for successful assembly to allow or disallow execution of the object program, then prints out any final error messages along with the flag legends. It also processes the compool for PUNCHC requests, writing it onto WK1.

ENTRY: This subroutine is entered at EOJ from the end of BLANALYZ or, if no symbolic analyzer request was made, at NOAN from the INIT4.

EXIT: This subroutine exits to SYSEOJ, or, if PUNCHC was requested, brings in a fresh copy of the BAL system record and returns to INIT1.

2.2 BLPAS1 ROUTINE

The BLPAS1 routine, the main body of the assembler first pass, consists of two sections: a mainline section and an interlude that terminates the first pass. The mainline section in turn divides into three routines: new statement, machine operation, and pseudo-operation. (See Figure 2-1.)

After initialization, the main section of BLPAS1 processes one source statement during each iteration. In the new statement routine, the statement is obtained in canonical form and its operation code is looked up to determine whether it is a pseudo-operation or a machine operation, thereby selecting which of two alternate routines is to be followed for its processing by BLPAS1.

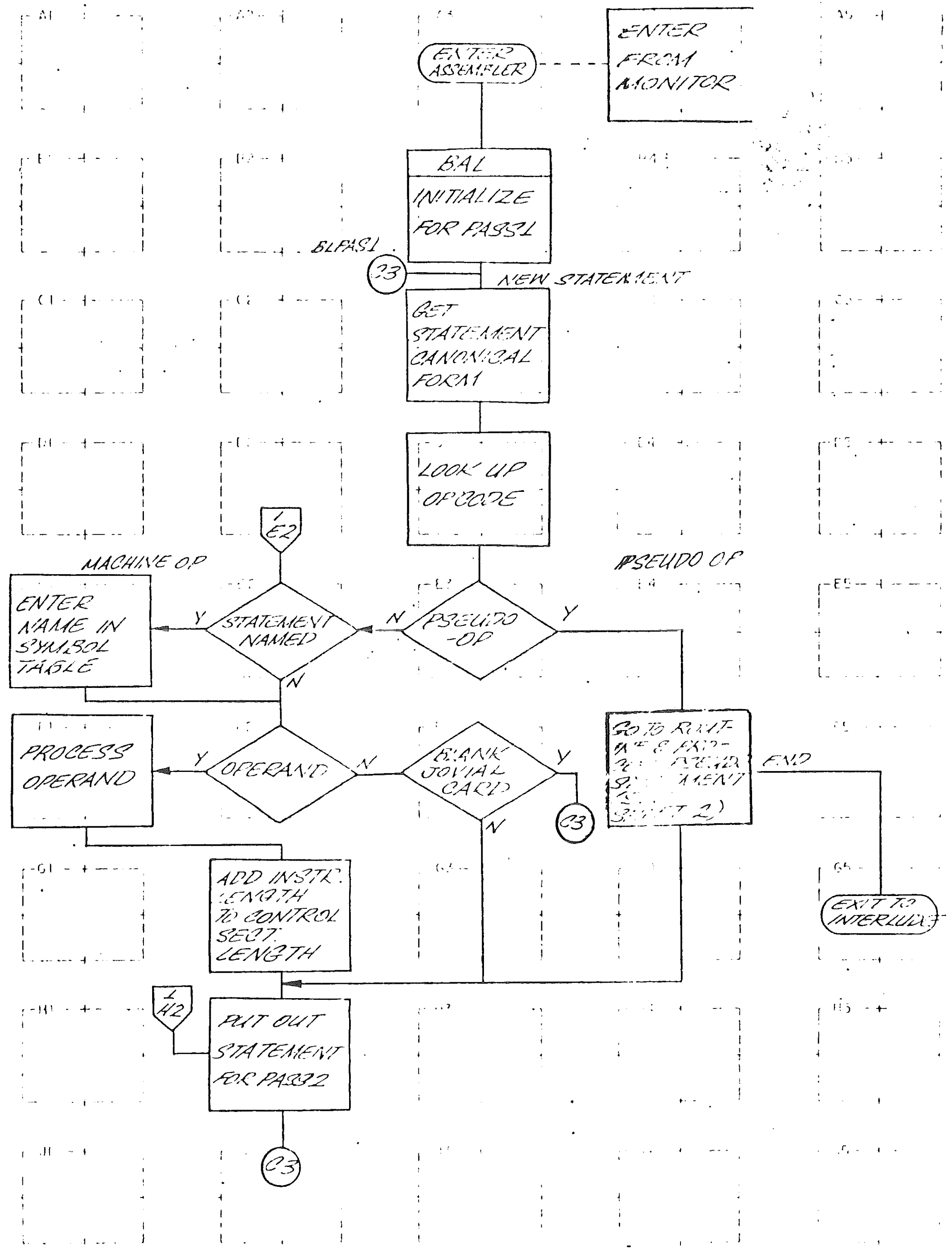
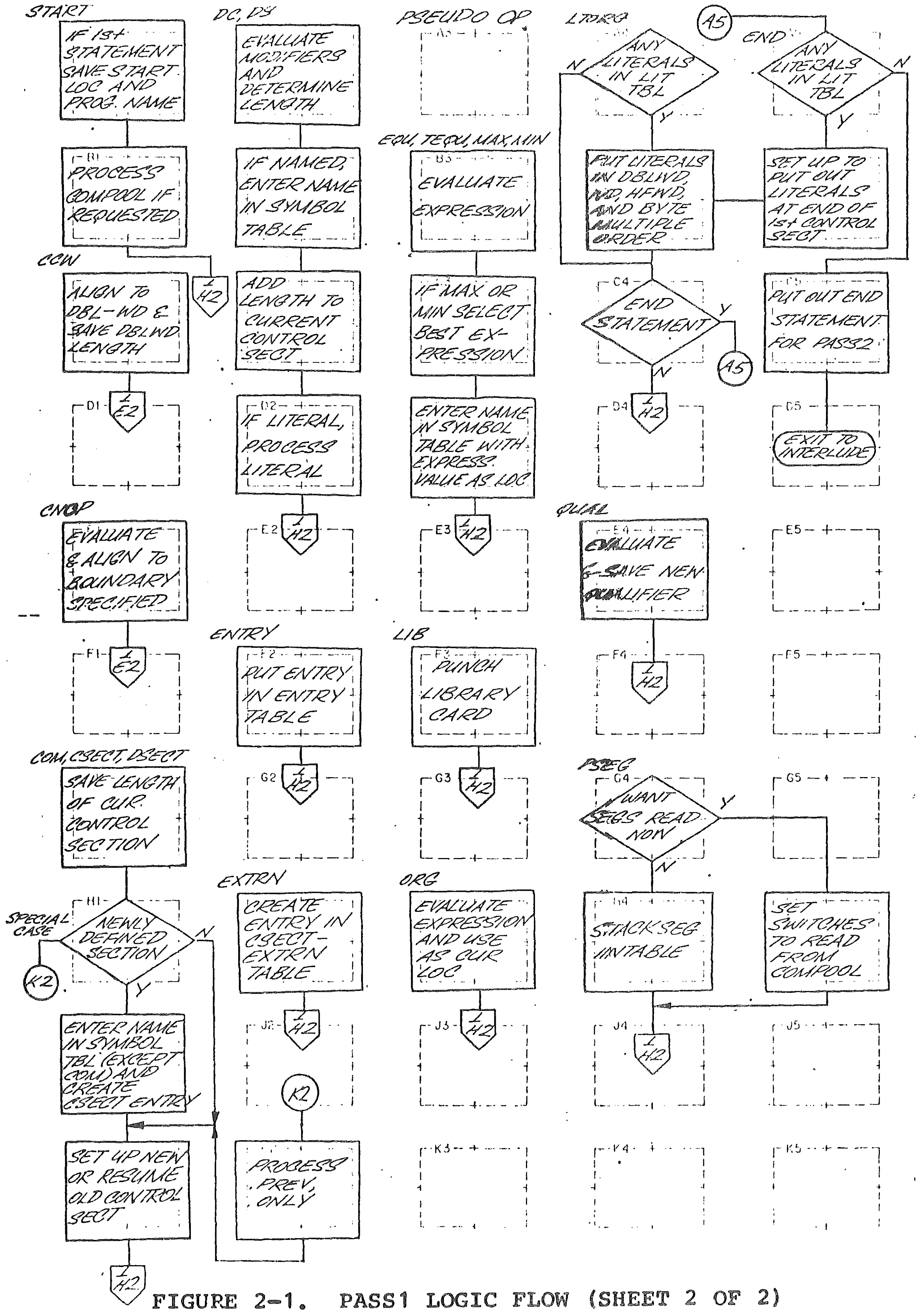


FIGURE 2-1. PASS1 LOGIC FLOW (SHEET 1 OF 2)



START
IF 1st STATEMENT SAVE START LOC AND PROG. NAME
B1
PROCESS COMPOOL IF REQUESTED
H2
CCW
ALIGN TO DBL-WD & SAVE DBLWD LENGTH
D1
E2
CNOP
EVALUATE & ALIGN TO BOUNDARY SPECIFIED
F1
E2
COM, CSECT, DSECT
SAVE LENGTH OF CUR CONTROL SECTION
H1
NEWLY DEFINED SECTION
SPECIAL CASE
K2
N
Y
ENTER NAME IN SYMBOL TBL (EXCEPT COM) AND CREATE CSECT ENTRY
SET UP NEW OR RESUME OLD CONTROL SECT
H2
DC, DS
EVALUATE MODIFIERS AND DETERMINE LENGTH
IF NAMED, ENTER NAME IN SYMBOL TABLE
ADD LENGTH TO CURRENT CONTROL SECT
D2
IF LITERAL, PROCESS LITERAL
E2
H2
ENTRY
F2
PUT ENTRY IN ENTRY TABLE
G2
H2
EXTRN
CREATE ENTRY IN CSECT-EXTRN TABLE
J2
H2
K2
PROCESS PREV. ONLY
PSEUDO OP
A3
EQU, TEQU, MAX, MIN
B3
EVALUATE EXPRESSION
IF MAX OR MIN SELECT BEST EXPRESSION
ENTER NAME IN SYMBOL TABLE WITH EXPRESS. VALUE AS LOC
E3
H2
LIB
F3
PUNCH LIBRARY CARD
G3
H2
ORG
EVALUATE EXPRESSION AND USE AS CUR LOC
J3
H2
K3
LTORG
ANY LITERALS IN LIT TBL
N
Y
PUT LITERALS IN DBLWD, WD, HFWD, AND BYTE MULTIPLE ORDER
C4
END STATEMENT
Y
A5
N
D4
H2
QUAL
E4
EVALUATE & SAVE NEW QUALIFIER
F4
H2
PSEG
G4
WANT SEGS READ NOW
Y
N
H4
STACK SEG IN TABLE
J4
H2
K4
A5
END
ANY LITERALS IN LIT TBL
N
Y
SET UP TO PUT OUT LITERALS AT END OF 1st CONTROL SECT
C5
PUT OUT END STATEMENT FOR PASS2
D5
EXIT TO INTERLUDE
E5
F5
G5
SET SWITCHES TO READ FROM COMPOOL
J5
K5

FIGURE 2-1. PASS1 LOGIC FLOW (SHEET 2 OF 2)

If the statement is a machine operation, the machine operation routine enteres the statement name (if any) in the symbol table and processes the operand, if present. The instruction length is then added to the accumulated program length and the statement is put out for pass 2. Return is then made to the new statement routine for the next source statement.

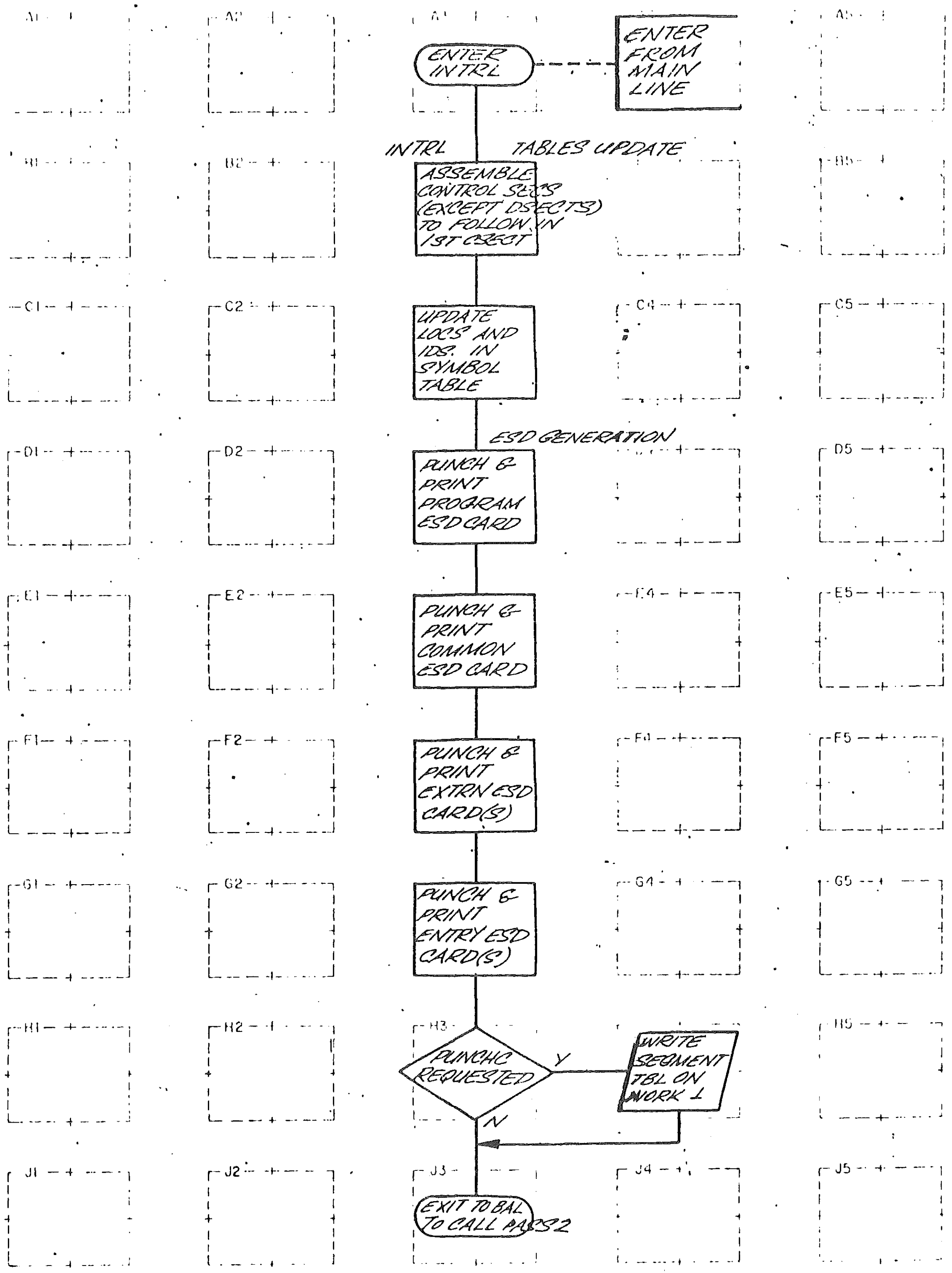
If the statement is a pseudo-operation, the appropriate processing subroutine is selected in the pseudo-operation routine and performed before the statement is put out for pass 2. The program then returns to the new statement routine for the next statement.

The pseudo-operation subroutines (shown in Figure 2-1) each process one type or several related types of pseduo-operation statements. Of particular note is the END statement, signaling the end of the source program. When encountered, the END statement causes any literals still in the literal table to be put out, followed by the END statement itself. The program then exits to the interlude section to terminate pass 1.

The interlude section of BLPAS1 (shown in Figure 2-2) divides into two routines: tables updating and ESD generation. In the tables updating routine, the control sections defined in the source program are assembled in the order of their definition into the first control section to produce a single relocatable block. The symbol locations in the symbol table are then updated, increasing each location by the new starting address of the control section in which the symbol was defined.

The ESD generation routine prepares ESD cards for the program, its common section, and for EXTRN and ENTRY symbols. If a listing was requested, the ESD deck is also listed. Similarly, if immediate execution was requested, ESD text is set up for loading after assembly is complete. Pass 1 is now completed, with BLPAS1 exiting to BAL to prepare for and call in BLPAS2.

Also included in BLPAS1 is a set of independent subroutines. These subroutines are described separately, following the description of the interlude.



ENTER INTRL
ENTER FROM MAIN LINE
INTRL
TABLES UPDATE
ASSEMBLE CONTROL SECTS (EXCEPT DSECTS) TO FOLLOW IN 1ST CSECT
UPDATE LOCS AND IDS. IN SYMBOL TABLE
ESD GENERATION
PUNCH & PRINT PROGRAM ESD CARD
PUNCH & PRINT COMMON ESD CARD
PUNCH & PRINT EXTRN ESD CARD(S)
PUNCH & PRINT ENTRY ESD CARD(S)
PUNCHC REQUESTED
Y
N
WRITE SEGMENT TBL ON WORK 1
EXIT TO BAL TO CALL PASS2

FIGURE 2-2. PASS1 INTERLUDE

2.2.1 PASS1 Mainline

2.2.1.1 Initialization Routine (PASS1)

FUNCTION: This routine (Chart AA4) initializes, on first entry only, for PASS1 and obtains the first source statement. If this statement is a comment, ICTL, JOV, or TITLE, it is processed and printed out and another statement is obtained. This process is repeated until a statement other than one of the above types is obtained.

ENTRY: This routine is entered at the first location of the PASS1 routine when control is turned over by the monitor.

EXIT: This routine exits to XXFST in the new statement routine.

2.2.1.2 New Statement Routine (BEGIN)

FUNCTION: This routine (Chart AA5) obtains a source statement and determines whether it is a machine operation or a pseudo-operation.

ENTRY: The normal entry for this routine is at BEGIN. On the first time through, it is entered at XXFST from the initialization routine of PASS1. It may be re-entered at AHEAD from the START subroutine if the START statement is missing on the first time through. It is also entered at OPERR if an incorrect operation code is encountered by the pseudo-operation routine.

OPERATION: This routine begins by calling IOGET1 to get a statement in canonical form. If the statement is a comment, it is put out by IOPUT1 and the routine restarts on the next statement.

On the first time through, the routine tests for a START statement. On all other iterations, the operation code is checked for valid length, then submitted for a search of the operation code table (see Subsection 4.1) by BLOPLKUP.

EXIT: If the operation code is a pseudo-operation, this routine exits to PSEUDO in the pseudo-operation routine. If the operation code is a machine operation, this routine exits to the machine operation routine.

ERRORS: If the first statement is not a START statement, diagnostic 71 is put out (see Subsection 6.4), the START subroutine is entered to force a START, and return is made to AHEAD to process the non-START first statement.

If a statement contains no operation code or one of excessive length, exit is made to OPERR in the machine operation routine, thereby ignoring the operand field.

2.2.1.3 Machine Operation Routine

FUNCTION: This routine (Charts AB, AC) reserves the object code length required for a statement, enteres the statement name (if present) in the symbol table, and validates the statement operand(s) (if present), then puts out the statement for pass 2.

ENTRY: This routine is normally entered by fall-through from the new statement routine for processing of a machine operation statement. It is also entered at OPL2 or OPL4 from some pseudo-operation subroutines to reserve space in the object program. It may be entered at OPERR if an invalid operation code is found by the new statement or pseudo-operation routine. Return from pseudo-operation subroutines is to SYMCK to pick up the statement name or to FINISH to put out the statement for pass 2.

OPERATION: On fall-through from the new statement-routine, this routine determines the instruction length and saves it for determination of program length. This routine next checks for a statement name, validates the symbol, calls BLSLKUP to search the symbol table (see Subsection 6.1) for that symbol and, if not found, enters the symbol in the symbol table.

If operand scan is not blocked (by an invalid operation code), the statement operand field is scanned. If a literal is found, the LITRTN subroutine is called to validate it and enter it in the literal table (see Subsection 6.1); since only one literal is allowed in a statement, return is to WINDUP to terminate the routine.

If the operand is not a literal, BLBRKUP is called to scan the expression and check its syntax. The expression terminator is then checked to determine further action (e.g., look for included literal, scan next operand); usually a partial or complete repetition of the operand scan.

The routine terminates by adding the saved instruction length to the current location counter, then putting out the source statement for pass 2.

EXIT: This routine returns to BEGIN in the new statement routine.

ERRORS: An invalid symbol in the statement name field is ignored. An invalid operand expression is similarly ignored, as is the rest of the operand field.

2.2.1.4 Pseudo-Operation Routine (PSEUDO)

FUNCTION: This routine (Chart AD) selects and uses the proper subroutine to process a pseudo-operation statement.

ENTRY: This routine is entered at PSEUDO from the new statement routine.

OPERATION: This routine uses a pointer in the operation code table to select the proper pseudo-operation subroutine. These subroutines are described separately later.

EXIT: This routine exits to the selected subroutine.

ERRORS: An invalid operation code mnemonic causes a return to OPERR in the machine operation routine.

COMMENTS: Certain statements are ignored by BLPAS2 (e.g., JOVIAL diagnostic, USING, and PRINT). For an EXBC or EXBCR (Extended Branch) statement, BLPAS1 simply saves the proper instruction length, then treats the statement as a machine operation, leaving interpretation of the extended mnemonic to BLPAS2.

2.2.1.5 DCL Subroutine

FUNCTION: This subroutine issues diagnostic 35 (see Subsection 6.4) when a DCL source statement is encountered.

ENTRY: This subroutine is entered at DCL from the pseudo-operation routine.

EXIT: This subroutine exits to DGCSEC in the CSECT subroutine to mark the statement to be ignored by pass 2.

COMMENTS: The fictitious operation code DCL is used by BLPAS1 to issue a literal. As such it is handled differently than an illegal operation, which it would be if encountered as a source statement.

2.2.1.6 CCW Subroutine

FUNCTION: This subroutine saves an aligned doubleword space for the Channel Command Word (CCW).

ENTRY: This subroutine is entered at CCW from the pseudo-operation routine.

EXIT: This subroutine returns to SYMCK in the machine operation routine to treat the statement as a machine operation.

2.2.1.7 LIB Subroutine

FUNCTION: This sequence (Chart AD) moves 24 characters or less from the operand of a LIB statement (arguments) into columns 19-42 of a card image and punches a LIB card.

ENTRY: This subroutine is entered at LIB from the pseudo-operation routine.

EXIT: This subroutine exits to FINISH in the machine operation routine.

2.2.1.8 QUAL Subroutine

FUNCTION: This sequence (Chart AD) validates a new qualifier and puts it in the communications region.

ENTRY: This subroutine is entered at QUAL from the pseudo-operation routine.

EXIT: This subroutine exits to FINISH in the machine operation routine.

ERRORS: If the qualifier is invalid, diagnostic 2 is issued, and the CNOP subroutine is entered at IGCNOP to mark the source statement to be ignored by pass 2.

2.2.1.9 START Subroutine

FUNCTION: This subroutine (Charts AE1, AE2) evaluates and saves the starting location to be assigned for the object program, then validates the program name and enters it in the symbol table, the punch area for the LIB cards, and the ENTRY table. It also checks for a compool request.

ENTRY: This subroutine is normally entered at START from the pseudo-operation routine. It is entered at BLANKS when the new statement routine finds that the first statement is not a START statement.

OPERATION: This subroutine begins by checking for a starting location. If present, the address is evaluated and saved. The name field is then checked for a symbol to be used as the program name. If present, the symbol is validated, BLSLKUP is called to search the symbol table, and the symbol is entered in the symbol table. (See Subsection 6.1.) If no program name appears in the statement, the subroutine is entered at BLANKS and the forced program name .NONAME is used. The supplied or forced program name is stored in the communications region and in the punch area for LIB cards, and is entered in the ENTRY table. The start card is also checked for a compool request. If one is present, it is validated. The routine then checks .LIB and/or .CMP for an MLC (Merged Library/Compool) or compool tape containing the desired compool, requests a new tape if necessary and positions the tape to the start of the requested compool.

EXIT: This subroutine exits to FINISH in the machine operation routine after it processes a START first statement or after it disposes of a non-first START statement. After entry at BLANKS to take care of a non-first START statement, the subroutine exits to AHEAD in the new statement routine to process the first statement.

ERRORS: A START statement after the first source statement causes diagnostic 15 to be issued; the statement is ignored.

If the START statement beginning the source program carries no starting location, diagnostic 8 is issued.

If the MEVAL subroutine finds an error in the expression for the starting location, it is ignored. Other faults in the starting location assignment cause it to be ignored and a diagnostic issued as follows.

Fault	Diagnostic
Not absolute	17
Not address type	31
Bad terminator	14

In addition, if the starting location assignment is not aligned to a doubleword boundary, it is adjusted and saved, and diagnostic 72 is issued.

If an invalid compool is requested, or the compool cannot be located, assembly is terminated.

COMMENTS: The MEVAL subroutine calls the BLEXVAL routine, making the output of BLEXVAL (Subsection 6.3) available to this subroutine.

2.2.1.10 CNOP Subroutine

FUNCTION: This subroutine (Chart AF) validates the operands of a Conditional No-Op (CNOP) pseudo-operation and, if valid, aligns the current location counter to the designated boundary.

ENTRY: This subroutine is entered at CNOP from the pseudo-operation routine. Some other pseudo subroutines use the error handling and return provisions of this subroutine, entering it at IGCNOP or at CNOPEX.

OPERATION: This subroutine evaluates the first operand and validates its value and terminator. The subroutine then evaluates the second operand and validates its value and terminator. The current location counter is then aligned to the boundary specified. Finally, the operand scan switch is turned off (because the operands have already been processed) and an instruction length of 0 is saved (because the location counter has already been incremented if required).

EXIT: This subroutine returns to SYMCK in the machine operation routine to process the statement name, if any.

ERRORS: For any operand error, the statement is marked to be ignored by pass 2 and no change is made to the location counter.

If either operand value is incorrect, diagnostic 31 is also issued.

If either terminator is incorrect, diagnostic 14 is also issued.

2.2.1.11 CSECT, DSECT Subroutine

FUNCTION: This subroutine (Charts AG1, AG2, AG3, AH) examines a CSECT or DSECT statement and either creates a new control section (by making an entry in the CSECT-EXTRN table) or resumes the already created control section renamed by this statement.

ENTRY: This subroutine is entered at CSECT or at DSECT from the pseudo-operation routine. Some other pseudo-subroutines use the error handling and return provisions of this subroutine, entering it at SMDIAG or at IGCSEC.

OPERATION: This subroutine begins, for either a CSECT or a DSECT statement, by checking for a symbol in the name field. An unnamed CSECT is rejected if the first control section is named; otherwise, it resumes the first control section. With the name validated (or its absence allowed), the current control-section data (e.g., current location) is moved to its CSECT-EXTRN table entry from the communications region in preparation for a change to a different current control section. Special (.PREV, .ONLY) CSECTs are also processed.

The first CSECT statement encountered causes a test for an implied unnamed-first control section; i.e., have statements already been assembled into a control section (the first and unnamed) that precedes the control section defined by the first CSECT statement? If not, this first CSECT statement defines the first control section.

Any succeeding CSECT or DSECT statement causes a search of the CSECT-EXTRN table. If a match is found, the control section named by the statement is resumed (e.g., CSECT location counter is loaded as the currect location counter). If no match is found, a new control section is created. A .PREV CSECT builds a CSECT card for the last previous CSECT and re-enters CSECT processing with it.

Creation of a new control section begins by entering the name in the symbol table. Then, the name, ID, and type are entered in the CSECT-EXTRN table. Finally, this data describing the newly created or resumed control section is transferred to the communications region to identify it as the current control section.

EXIT: This subroutine returns to FINISH in the machine operation routine.

ERRORS: If a DSECT statement is unnamed, or CSECT statement is unnamed and the first control section is named, diagnostic 8 is issued and the statement is marked to be ignored by pass 2.

If a match is found between a CSECT or DSECT statement name and a non-EXTRN entry in the CSECT table (CSECT statement naming a dummy control section or DSECT statement naming a control section), diagnostic 68 is issued and the named control section is resumed.

If a new CSECT statement name has already been defined (i.e., multi-defined symbol), the name is blanked and the CSECT-EXTRN table searched again. The CSECT statement will thus resume the first control section if that is unnamed or will create a new unnamed control section; this control section can be resumed but two such multi-defined control sections will be assembled together.

If the name of the first DSECT statement is multi-defined, and it was not defined by an EXTRN, its name is changed to . (period) and entered in the CSECT-EXTRN table. All succeeding DSECT statements with multi-defined names are ignored. The DSECT names defined by an EXTRN are allowed and taken as a DSECT definition (or resumption).

The .OVLY CSECT without both ''V'' and ''POOL'' on the Start card, and .PREV CSECT not in a DSECT, are both treated as errors.

2.2.1.12 COM Subroutine

FUNCTION: This subroutine (Chart AH) initiates or resumes assembly in the common section of the program in response to a COM statement.

ENTRY: This subroutine is entered at COM from the pseudo-operation routine.

OPERATION: This subroutine begins by checking for the absence of a name in the COM statement. The current location counter is stored in the current CSECT-EXTRN table entry and the common entry in that table is initialized if not already done. The subroutine then enters the CSECT, DSECT subroutine to set up the common section as the current assembly section.

EXIT: This subroutine returns to FINISH in the machine operation routine to put out the statement.

ERRORS: If the COM statement is named, diagnostic 67 is issued, the rest of the subroutine is bypassed, and entry is made at IGCSEC in the CSECT, DSECT subroutine to mark the statement to be ignored by pass2.

2.2.1.13 DCDS Subroutine

FUNCTION: This subroutine (Chart AI) evaluates the operand of a DC or DS statement and adds its length to the current location counter, enters the statement name, if any, in the symbol table, and processes a literal, if present.

ENTRY: This subroutine is entered at DCDS from the pseudo-operation routine.

OPERATION: This subroutine calls BLCONMOD to evaluate any modifiers the operand may carry, aligns the location counter to the proper boundary for the operand, then checks for a statement name. If present, the symbol is entered in the symbol table, using the attributes developed by BLCONMOD. The location counter is incremented by the length of the constant. If the operand contains an address literal, it is processed by Subroutine LITRTN.

EXIT: This subroutine returns to FINISH in the machine operation routine.

ERRORS. If the modifiers contain errors, diagnostics provided by BLCONMOD are issued, the statement processing is bypassed, and entry is made at IGCNOP in the CNOP subroutine. As a result, the statement is marked to be ignored by pass 2, the operand scan is suppressed, and return is made to SYMCK in the machine operation routine (for symbol check) with a saved instruction length of 0.

2.2.1.14 ENTRY Subroutine

FUNCTION: This subroutine (Chart AJ) enters the symbol(s) in the ENTRY statement operand field in the ENTRY table, first searching the table to prevent duplication.

ENTRY: This subroutine is entered at ENTRY from the pseudo-operation routine.

OPERATION: This subroutine validates the first operand expression and, if valid, searches the ENTRY table for a duplicate. If none is found and the expression terminator is valid, the symbol is entered in the ENTRY table. The process is repeated for the next expression, and so on.

EXIT: This subroutine returns to FINISH in the machine operation routine to put out the statement.

ERRORS: If an expression is invalid, that symbol is ignored. If an expression is invalid and its terminator is missing (invalid), the subroutine exits without entering any following symbols from that statement into the ENTRY table.

If an ENTRY symbol duplicates one already entered, diagnostic 69 is issued and the next expression, if any, is processed.

If the ENTRY table is filled, diagnostic 76 is issued and the symbol that would have caused overflow is not entered.

2.2.1.15 EQMAMI, TEQU Subroutine

FUNCTION: This subroutine (Chart AK) enters the statement name in the symbol table with its location derived from the expression in the operand field, first validating the symbol and the operand expression(s).

ENTRY: This subroutine is entered, from the pseudo-operation routine, at TEQU for a TEQU statement or at EQMAMI for an EQU, MAX, or MIN statement.

OPERATION: This subroutine, if entered at TEQU, treats the TEQU operation code as an EQU by ignoring the first character. Presence of the statement name is then verified and the symbol is validated. The first operand (or only one) is then evaluated.

For EQU or TEQU, the statement name is entered in the symbol table (this may simply change the location attribute if a TEQU statement names an already defined symbol).

For MAX or MIN, a second operand is checked for and evaluated, then compared with the location attribute of the first operand. The better attribute is saved and, if there is another operand, the process is repeated before an entry is made in the symbol table.

EXIT: This subroutine returnts to FINISH in the machine operation routine to put out the statement unless an error is found.

ERRORS: If the statement is unnamed, processing is bypassed and exit is made to SMDIAG in the CSECT, DSECT subroutine to issue diagnostic 8 and mark the statement to be ignored by pass 2.

If the statement name or the first operand is invalid, processing is bypassed and exit is made to IGCSEC in the CSECT, DSECT subroutine to mark the statement to be ignored by pass 2.

If the first operand is mispunctuated or too long, diagnostic 4 is issued, processing is bypassed, and the statement is marked to be ignored by pass 2 before the normal exit is taken.

If the last operand terminator is not a blank, diagnostic 14 is issued, no symbol table entry is made, and the statement is marked to be ignored by pass 2 before the normal exit is taken.

For MAX or MIN, an invalid operand expression prevents entry in the symbol table and causes the statement to be marked to be ignored by pass 2 before the normal exit is taken.

For MAX or MIN, reference to operand symbols defined in different control sections causes diagnostic 66 to be issued; no entry is made in the symbol table and the statement is marked to be ignored by pass 2 before the normal exit is taken.

2.2.1.16 EXTRN Subroutine

FUNCTION: This subroutine (Chart AL) validates the operand(s) of an EXTRN statement and, if they have not been previously defined and do not duplicate other EXTRN entries in the CSECT-EXTRN table, enters these symbols in the symbol table and the CSECT-EXTRN table.

ENTRY: This subroutine is entered at EXTRN from the pseudo-operation routine.

OPERATION: This subroutine first validates an operand, then searches the CSECT-EXTRN table for possible duplication. If no duplicate is found and the operand terminator is valid, the operand symbol is entered in the symbol table with a new ID to identify it as an EXTRN. This attempt at entry in the symbol table may return with an indication that the symbol has already been defined in the BAL program. If the symbol has not already been defined, it is entered in the CSECT-EXTRN table. The process repeats for any additional operand(s).

EXIT: This subroutine returns to FINISH in the machine operation routine to put out the statement.

ERRORS: An invalid expression is ignored; if its terminator is missing, any following operands are ignored also.

A duplicate EXTRN causes diagnostic 70 to be issued; any following operands are processed.

A bad expression terminator causes diagnostic 14 to be issued; any following operand is ignored.

An already defined symbol in an EXTRN statement is ignored.

2.2.1.17 LTORG Subroutine

FUNCTION: This subroutine (Charts AM, AN) (called either by an LTORG statement or indirectly by an END statement) puts out all literals in the literal table as DCL statements, first putting out doubleword multiples, then fullword multiples, then halfword multiples, and finally odd-byte length multiples.

ENTRY: This subroutine is entered at LTORG from the pseudo-operation routine. It is also entered at LITPUT from the END subroutine.

OPERATION: If the LTORG statement is named, the symbol is entered in the symbol table and the LTORG statement is put out. A check for literals in the literal table now either causes return to the new statement routine for the next source statement or the putting out of the literals in the table.

If there are literals in the table, the subroutine sets up to search the table for literals of doubleword multiple length. When one is found, its dummy name is entered in the symbol table, its length is added to the current location counter, and it is put out in a DCL statement. After the literal table has been searched for doubleword multiples, the process is repeated for fullword multiples, then for halfword multiples, and finally for all other literals (odd-byte length).

After all literals have been put out, the subroutine initializes for buildup of a new literal table.

EXIT: This subroutine either returns to BEGIN in the new statement routine or, if entered from the END subroutine, returns to that subroutine.

ERRORS: If a dummy name cannot be entered in the symbol table because of table overflow, diagnostic 79 is issued.

2.2.1.18 ORG Subroutine

FUNCTION: This subroutine (Charts AO1, AO2, AO3) validates the operand of the ORG statement and, if valid, sets the current location counter to its value.

ENTRY: This subroutine is entered at ORG from the pseudo-operation routine.

OPERATION: This subroutine begins by evaluating the ORG expression. A sequence of tests then validates the expression before its value is placed in the current location counter.

EXIT: This subroutine returns to CNOPEX in the CNOP subroutine to block operand scan and save an instruction length of 0 before going to SYMCK in the machine operation routine to process the statement name, if any.

ERRORS: For any of the following errors, the current location counter is unchanged and the ORG statement is marked to be ignored by pass 2. (Where diagnostics are issued, they are noted.)

a. Errors in expression.

b. Symbol in expression is defined in another control section - diagnostic 66.

c. Expression is not an address constant - diagnostic 31.

d. Expression terminator invalid - diagnostic 14.

e. Expression too long or mispunctuated - diagnostic 4.

2.2.1.19 PSEG Subroutine

FUNCTION: This subroutine validates the operands of PSEG cards, and initiates reading of the compool when requested.

ENTRY: This subroutine is entered at PSEG from the pseudo-operation routine, or at PSOUT10 when the END card requires a compool read.

OPERATION: This subroutine first checks the operand field for segment names and/or special codes. Segment names are placed in the segment table. The .LIST and .PUNC options cause appropriate switches to be set. The .ALL option causes the entire compool to be read in, while .USE searches the segment table to assure at least one valid request, and the reads in all those segments in the table with a status of ''WAITING.'' The read is accomplished by setting switch GPOOL to cause the RDMRG routine to read from compool, and then returning to BEGIN to get another input record. Unless LISTD or .LIST have been specified, an NLIST card is generated for BLPAS2 immediately before exiting.

EXIT: This routine exits to BEGIN in the new statement routine.

ERRORS: This routine will issue a serious diagnostic on an invalid PSEG or segment table overflow, and a warning on a duplicate PSEG request. Diagnostics may appear with the PSEG ,.USE card rather than the actual PSEG name card which caused the error.

2.2.1.20 END Subroutine

FUNCTION: This subroutine (Charts A01, A02, A03) or recognizing the end of the source program, brings in any outstanding compool segments requested, puts out literals at the end of the first control section, and exits to the interlude.

ENTRY: This subroutine is entered at END from the pseudo-operation routine, and is re-entered from the LTORG subroutine after it has called that subroutine. It is re-entered if compool segments have been brought in at this point.

OPERATION: When the subroutine is first entered, it checks for any outstanding compool requests. If found, it stacks the END card and goes to read the compool. It then checks whether there are literals in the literal table. If so, the subroutine sets up a control setting with a start address immediately following the high location of the first control section. After aligning the current location counter to a doubleword boundary, it enters the LTORG subroutine at LITPUT. The LTORG subroutine returns, after all literals have been put out, to END. The current location counter, incremented by the length of the literals, is stored in the first CSECT-EXTRN table entry and the END statement is then put out.

EXIT: This subroutine exits to INTRL in the interlude section of PASS1.

ERORRS: Compool errors are given under the PSEG routine.

2.2.2 PASS1 Interlude

The interlude section of PASS1 is entered after the END statement signals the end of the source program. As a preliminary, the interlude rewinds .WORK2, thereby making the overflow of the intermediate buffer available to BLPAS2 by the time it finishes the contents of the intermediate buffer. The interlude then proceeds (as shown in Figure 2-2) through table updating and ESD generation, concluding BLPAS1. Exit is made to INIT3 in BAL to initialize for and bring in BLPAS2.

Table updating is performed by two subroutines: CSECT table update and symbol table update.

The ESD generation is performed by four subroutines:

1. Program ESD

2. Common ESD

3. EXTRN ESD

4. ENTRY ESD

2.2.2.1 CSECT Table Updating Subroutine

FUNCTION: This subroutine (Chart AP) assembles all control sections into a single control section and determines the object program length.

ENTRY: The subroutine is entered at INTRL from PUTEND in the END subroutine of BLPAS1.

OPERATION: This subroutine uses the current-location-counter field as the program location counter, loading it with the location specified by the source program START statement. The subroutine then scans the CSECT table for a control section. If a DSECT, EXTRN, or COM entry is found, its location counter is zeroed.

When the first control section is found, the program location counter is set as its starting address and its length is added to the counter, rounded up to the next doubleword boundary. The process is repeated for the second control section, assigning it an aligned starting address following the end of the first control section. In addition, the control section ID is changed to that of the first control section. This process continues until all control sections have been assembled into the first. The program length is then computed and the subroutine ended.

EXIT: This subroutine exits to the symbol table updating subroutine.

2.2.2.2 Symbol Table Updating Subroutine

FUNCTION: This subroutine (Chart AP) increases each symbol location by the starting address of the control section in which it was defined and changes the symbol ID to that of the first control section.

ENTRY: This subroutine is entered from the CSECT table updating subroutine.

EXIT: This subroutine exits to the Program ESD Subroutine in the ESD generation routine.

2.2.2.3 Program ESD Subroutine

FUNCTION: This subroutine (Chart AQ) initializes for ESD generation and puts out the program ESD card; the program ESD is listed if listing is requested.

ENTRY: This subroutine is entered from the symbol table update subroutine.

EXIT: This subroutine exits to the common ESD subroutine.

2.2.2.4 Common ESD Subroutine

FUNCTION: This subroutine (Chart AQ) determines the length of the common section, if any, and puts out a common ESD card; the common ESD is listed, if listing is requested.

ENTRY: This subroutine is entered from the program ESD subroutine.

EXIT: This subroutine exits to I3EXT in the EXTRN ESD Subroutine.

2.2.2.5 EXTRN ESD Subroutine

FUNCTION: This subroutine (Chart AQ) scans the CSECT-EXTRN table for EXTRN entries, putting up to three consecutive EXTRN symbols on one ESD card; each EXTRN is also listed if listing is requested.

ENTRY: This subroutine is entered at I3EXT from the common ESD subroutine.

EXIT: This subroutine exits to I3ENT in the ENTRY ESD subroutine.

2.2.2.6 ENTRY ESD Subroutine

FUNCTION: This subroutine (Chart AR) puts out ENTRY ESD cards with up to three symbols on each card; if listing is requested, the ENTRY ESDs are also printed.

ENTRY: The subroutine is entered at I3ENT from the EXTRN ESD Subroutine.

OPERATION: This subroutine scans the ENTRY table and checks that the ENTRY symbol has been entered in the symbol table. It then checks that the symbol was defined in a control section or in the common section, then moves the symbol with its ESD-ID to the card image. An ESD line is printed if listing is requested. When the card image is filled (three ENTRYs), the card is put out.

When the end of the ENTRY table is reached, the last ESD line is printed (if listing is requested) and the card image is punched even if not full. Upon completion of this subroutine, BLPAS1 program terminates.

EXIT: This subroutine exits to INIT3 in BAL.

2.2.3 BLPAS1 Common Subroutines

Included within the BLPAS1 program are 14 subroutines that are called to perform various operations for the main body of the program. These subroutines, listed below, are each described separately.

1. MPUTS - Put a statement record
2. MPUTD - Put a diagnostic record
3. HFWRD - Align current location counter to a halfword boundary
4. DBLWRD - Align current location counter to a doubleword boundary
5. STFULL - Process symbol table overflow
6. CTFULL - Process CSECT-EXTRN table overflow
7. L1TRTN - Process literal
8. SYMVAL - Validate a statement symbol
9. STNTRY - Create a symbol table entry
10. MEVAL - Call BLEXVAL and process diagnostics
11. HEX2CM - Convert hexadecimal number into EBCDIC characters
12. MBRKUP - Call BLBRKUP and evaluate expression
13. PUTESD - Put an ESD card
14. PRTESD - Print ESD line

2.2.3.1 MPUTS Subroutine

FUNCTION: This subroutine (Chart AS) puts out a source statement for pass 2.

ENTRY: This subroutine is entered at MPUTS from the machine operation routine or from the END or LTORG subroutine, with the address of the source statement in a general register.

OPERATION: This subroutine calls IOPUT1 to put out the source statement. If the ignore switch is on, the statement is prefixed with an ignore code so that pass 2 will ignore it. (See Subsection 6.2 for format.)

EXIT: This subroutine exits to the calling point.

2.2.3.2 MPUTD Subroutine

FUNCTION: This subroutine puts out a diagnostic record for pass 2.

ENTRY: This subroutine is entered at MPUTD from the new statement routine or from those pseudo-operation or BLPAS1 subroutines that issue diagnostics.

OPERATION: This subroutine calls IOPUT1 to put out the diagnostic record. Subsection 6.2 gives the format of the diagnostic record built at DIAG.

EXIT: This subroutine exits to the calling point.

2.2.3.3 HFWRD Subroutine

FUNCTION: This subroutine aligns the current location counter to a halfword boundary if it is not so aligned.

ENTRY: This subroutine is entered at HFWRD from the machine operation routine.

EXIT: This routine exits to the calling point.

2.2.3.4 DBLWRD Subroutine

FUNCTION: This subroutine aligns the current location counter to a doubleword boundary if it is not so aligned.

ENTRY: This subroutine is entered at DBLWRD from the CCW, END, LTORG, START, CSECT table update, or common ESD subroutine.

EXIT: This subroutine exits to the calling point.

2.2.3.5 STFULL Subroutine

FUNCTION: This subroutine processes an overflow of the symbol table.

ENTRY: This subroutine is entered at STFULL from the machine operation routine or from the DCDS, EQMAMI, EXTRN, LTORG, START, or CSECT, DSECT subroutines.

OPERATION: This subroutine puts out diagnostic 43 each time a symbol table overflow condition is encountered.

EXIT: This subroutine exits to the calling point.

2.2.3.6 CTFULL Subroutine

FUNCTION: This subroutine (Chart AS) processes an overflow of the CSECT-EXTRN table.

ENTRY: This subroutine is entered at CTFULL from the CSECT, DSECT or EXTRN subroutine.

OPERATION: This subroutine puts out diagnostic 74 and deletes the CSECT name from the symbol table whenever that entry will overflow the CSECT-EXTRN table.

EXIT: This subroutine exits to the calling point.

2.2.3.7 LITRTN Subroutine

FUNCTION: This subroutine (Chart AT) validates the literal and its modifiers, searches the literal table for a duplicate, and enters it if no duplicate is found. Finally, the subroutine puts out a literal reference record for pass 2.

ENTRY: This subroutine is entered at LITRTN from the machine operation routine or the DCDS subroutine, with the address of the literal in a general register.

OPERATION: This subroutine calls BLCONMOD to evaluate the literal modifiers, then validates the literal terminator, type, and length. Next, the literal table is searched for an identical literal. If found, the dummy tag of the table entry and the literal source length are set up in a literal reference record at LIT (see Subsection 6.2) and IOPUT1 is called to put out the record.

If no duplicate of the literal is found in the literal table, the literal is entered in the table and the literal reference record put out as before.

EXIT: This subroutine returns to the calling point.

ERRORS: An invalidity in the literal bypasses the issuance of a literal reference record or entry in the literal table.

If the literal table is filled, a new literal is put out in a literal reference record with a zero dummy tab, preceded by diagnostic 75.

2.2.3.8 SYMVAL Subroutine

FUNCTION: This subroutine (Chart AU) validates the symbol submitted to it and appends the current qualifier.

ENTRY: This subroutine is entered at SYMVAL from the machine operation routine or from the DCDS, EQMAMI, LTORG, START, or CSECT, DSECT subroutine.

OPERATION: This subroutine checks the symbol for length, valid first character, and valid succeeding characters. If validated, the current qualifier is added to the symbol.

EXIT: This subroutine returns to the calling point if the symbol is not valid. It returns to the second instruction at the calling point if the symbol is valid.

ERRORS: If the symbol length is excessive, diagnostic 3 is issued and return is made to the error return point in the calling routine.

If the first or succeeding characters are not valid, diagnostic 2 is issued and return is made to the error return point in the calling routine.

2.2.3.9 STNTRY Subroutine

FUNCTION: This subroutine (Chart AU) enters a symbol in the symbol table at the entry point found for it by the SLKUP routine.

ENTRY: This subroutine is entered at STNTRY from the machine operation routine or from the DCDS, EQMAMI, EXTRN, LTORG, START, or CSECT, DSECT subroutine.

OPERATION: This subroutine begins by checking whether the symbol has already been defined. If not, the symbol table entry is created and entered in the table.

EXIT: This subroutine always exits to the calling point.

ERRORS: If the symbol has already been defined, the table-entry control byte is changed to 'multi-defined' and diagnostic 12 is issued before return is made to the calling routine.

2.2.3.10 MEVAL Subroutine

FUNCTION: This subroutine (Chart AV) evaluates an expression and issues any diagnostics required.

ENTRY: This subroutine is entered at MEVAL from the CNOP, EQMAMI, ORG, or START subroutine, with the address of the expression in a general register.

OPERATION: This subroutine calls the BLEXVAL routine to evaluate the expression, then issues diagnostics for any errors found by BLEXVAL. If no fatal error (i.e., multi-defined symbol) is found, the subroutine checks the relocatability of the expression. If the expression is in the operand of a CNOP statement, the expression is checked to determine that it is absolute and within a maximum value.

EXIT: This subroutine returns to the calling point if the expression is not valid. It returns to the second instruction at the calling point if the expression is valid.

ERRORS: Errors found by BLEXVAL are issued as diagnostics and, if the errors are fatal, exit is made to the error return point in the calling routine.

If the expression is invalidly relocatable, diagnostic 7 is issued before return is made to the error return point.

For the operand of a CNOP statement, if the expression is not absolute, diagnostic 17 is issued and return is made to IGCNOP in the CNOP subroutine to mark the source statement to be ignored by pass 2.

For the operand of a CNOP statement, if the expression value is not less than 256, return is made to BADVAL in the CNOP subroutine to issue diagnostic 31 and mark the source statement to be ignored by pass 2.

2.2.3.11 HEX2CM Subroutine

FUNCTION: This subroutine (Chart AV) converts a hexadecimal number into EBCDIC characters for printing or punching in source program language.

ENTRY: This subroutine is entered at HEX3CM from the END or PUTESD subroutine, with the number to be converted in a general register, and the address of the output area in another.

EXIT: This subroutine exits to the calling point.

2.2.3.12 MBRKUP Subroutine

FUNCTION: This subroutine (Chart AW) isolates an expression within an operand, then validates the expression.

ENTRY: This subroutine is entered at MBRKUP from the ENTRY or EXTRN subroutine, with the address of the expression in a general register.

OPERATION: This subroutine calls the BLBRKUP routine and checks for any error that the routine may find in the expression. The expression is then validated and set up for delivery to the calling routine with the current qualifier appended or with the qualifier designated in the expression.

EXIT: This subroutine exits to the calling point if the expression is invalid. It returns to the second instruction at the calling point if the expression is valid.

ERRORS: If BLBRKUP finds an error in an EXTRN expression, the BLBRKUP diagnostic is issued before the subroutine returns to the error return point.

If BLBRKUP finds an error in an ENTRY expression, return is made to the error return point.

If the expression is too long, diagnostic 4 is issued before return is made to the error return point.

If the first character of the expression is invalid, or the qualifier element begins with other than a period, or there is a third element, diagnostic 31 is issued before return is made to the error return point.

2.2.3.13 PUTESD Subroutine

FUNCTION: This subroutine (Chart AX) punches an ESD card and initializes the card image for the next card.

ENTRY: This subroutine is entered at PUTESD from the EXTRN ESD or ENTRY ESD subroutine.

EXIT: This subroutine exits to the calling point.

2.2.3.14 PRTESD Subroutine

FUNCTION: This subroutine (Chart AX) prints an ESD line and sets up the next one for printing if a listing is requested.

ENTRY: This subroutine is normally entered at PRTESD from the common ESD, EXTRN ESD, or ENTRY ESD subroutine. On the first time through only, it is entered at EJECT from the Program ESD Subroutine if a listing is requested.

OPERATION: This subroutine is entered the first time to eject the listing page, print the ESD header, then set up the Program ESD line before returning to the calling routine.

On subsequent entries, this subroutine first checks whether a listing is requested. If so, it prints the line set up on its previous call and forces a page eject if the page is full before it sets up the next ESD line.

EXIT: This subroutine exits to the calling point, immediately if no listing is requested, or after setting up the next line on all but its last call. On its last call, exit is taken after printing the previously set-up line.

2.2.4 Get A Statement During PASS1 Routine (IOGET1)

FUNCTION: This routine (Chart CA) gets a statement during pass 1 and puts it into canonical form.

ENTRY: This routine is entered at IOGET1. It is called by BLPAS1.

OPERATION: This routine calls RDMRG to get a source statement, then determines whether the statement is a comment or not. A comment card is immediately moved to the output area with the comment field left-adjusted and the sequence field right-adjusted in the 80-column field.

If the statement is not a comment card, the symbol (if found) is moved to the output area, its length is computed and moved to the output, and a search is begun for the operation code. If no operation code is found, the statement is printed and ignored, and the routine calls RDMRG to get a new statement.

If an operation code is found, it is moved to the output area, its length is computed and moved to the output, and a search is begun for the operand.

If an operand is found, its length is determined by finding its rightmost non-blank character. The operand is then moved to the output area, followed by a single blank, and its length (with the blank) is also moved to the output. If no operand is found, only the blank is moved to the output area. Finally, the sequence field is moved to the output area and the total statement length is computed and moved to the output.

EXIT: This routine exits to the calling point.

ERRORS: If a BAL program on .INPUT lacks an END statement, the EOF return from the call of RDMRG forces the delivery of an END statement by IOGET1. A message in this statement's comment field describes the action taken.

2.2.5 Read And Merge Source Statement Routine (RDMRG)

The Read And Merge Source Statement Routine (RDMRG) is called by IOGET1 during pass 1 to read in a new statement in BAL language. These source statements may come from one of three sources:

1. .WORK1 if a compiled JOVIAL program is submitted, or if compool PUNCHC processing is being done.
2. .INPUT if a BAL program is submitted.
3. .LIB or .CMP if a compool is requested.

The first pass through RDMRG selects WORK1 or INPUT for all succeeding passes.

The RDMRG routine divides functionally into three subroutines and one subordinate subroutine as follows:

a. Subroutines

    1. Main Subroutine: This subroutine obtains a statement from the selected source and delivers it, then exits to IOGET1.

    2. First Pass Subroutine: This subroutine selects the source in accordance with control card information, then enteres the appropriate subroutine to deliver the first program statement.

    3. Compool Read Subroutine: This subroutine reads input from the compool.

b. Subordinate Subroutine

READ - Selects source on first pass and reads a statement.

2.2.5.1 RDMRG (Main) Subroutine

FUNCTION: This subroutine (Chart DA) obtains a statement from the source selected during the first pass and moves it to the delivery area before returning control to IOGET1.

ENTRY: The entry point for this subroutine is RDMRG. It is always entered from IOGET1 which provides the delivery address and .INPUT EOF return address out-of-line from the calling point.

OPERATION: On each pass, this subroutine begins by picking up the two parameters supplied by IOGET1. On the first pass, the main subroutine branches to the first pass subroutine for source selection. On all succeeding passes, it goes to the return point set on the preceding pass. If .WORK1 or .INPUT is the sole source, the main subroutine obtains a statement on each pass, moves it to the delivery area, and returns control to IOGET1. If the GPOOL switch is set, the main subroutine branches to PREAD to read from compool.

EXIT: This subroutine exits to the calling point in IOGET1.

2.2.5.2 ENTER (First Pass) Subroutine

FUNCTION: This subroutine (Chart DA) uses control card information to select the source for statements, then exits to the appropriate processing subroutine for the selected source.

ENTRY: This subroutine is entered at ENTER on the first pass through the main subroutine.

OPERATION: Upon entry into the first pass subroutine, the READ subroutine selects either .WORK1 (for JOVIAL compiler output) or .INPUT and reads the first statement from that source.

EXIT: This subroutine returns to the main subroutine to move the statement to the delivery area.

COMMENTS: The READ subroutine latches to its first selection and reads only from the selected source (.WORK1 or .INPUT) on subsequent calls.

2.2.5.3 Compool Read Subroutine (PREAD)

FUNCTION: This subroutine (Charts DB, DC) reads card images from the compool, either returning the card read to the caller, or selecting some other special card to be returned.

ENTRY: This routine is entered at PREAD from RDMRG if switch GPOOL is set to 1, or at UNSTND if switch GPOOL is set to 2.

OPERATION: If switches SENDS or SENDNX are set, a DSECT or a saved card are returned to the caller. On entry at UNSTND, a saved END card is returned to the caller. Otherwise a card is read from the compool and examined.

An END card terminates processing of the current segment, and returns to read another card.

An EXTRN terminates all compool reading. Diagnostics may be issued. If there is a stacked END card, GPOOL is set to 2. A .PREV CSECT is returned to the caller unless an overlay assembly is in process, in which case a comment is returned.

If a START card is read, the segment name is examined to see if it is wanted. If it is not, switch THISG is turned off, and cards are read to skip to the end of the segment. If the segment is needed, another card is read. If it is ENTRY Poolname, it is ignored, while anything else is saved. Switches are set to return a DSECT and the saved card if any, to the caller, on subsegment calls. An EXTRN is then returned to the caller.

EXIT: This subroutine returns to the calling point.

ERRORS: If there are any unsatisfied requests in the table at the end of the compool, diagnostics are issued.

2.2.5.4 READ Subroutine

FUNCTION: This subroutine (Chart DD), on the first pass, selects the source for statements for RDMRG and reads the first statement from that source. On each subsequent pass, this routine reads a statement from that source.

ENTRY: This subroutine is entered at READ from all three of the RDMRG subroutines.

OPERATION: On the first pass, this subroutine checks whether input is on .WORK1. If so, it latches to that source, rewinds it, and reads the first statement. If not, it latches to .INPUT and reads the first statement from there. On each subsequent pass, this subroutine reads one statement from the selected source.

EXIT: This subroutine normally exits after each pass to the calling point in RDMRG. An EOF causes return to the NOMORE subroutine in IOGET1.

2.2.6 Put Out A Statement During PASS1 Routine (IOPUT1)

FUNCTION: This routine (Chart NA) is used by BLPAS1 to put a statement into the intermediate buffer until it is filled, afterward onto .WORK2 if it is available.

ENTRY: This routine is entered at IOPUT1 from BLPAS1.

OPERATION: This routine first determines the record length from its type, then checks whether the intermediate buffer is filled. If not, the buffer is tested to see whether entry of this record will make the buffer overflow. If not, the record is entered in the buffer and its new entry point is saved. If entry would cause overflow, the buffer is marked as filled and the record is written insteand on .WORK2 if it is available. If .WORK2 is not available and a new entry will cause overflow, the assembly is terminated and a message indicating which card caused overflow is printed. If .WORK2 is avaiable all succeeding records are similarly written on .WORK2.

EXIT: This routine exits to the calling point in BLPAS1.

2.3 OPERATION CODE LOOKUP ROUTINE (BLOPLKUP)

FUNCTION: This routine (Chart EA) searches the operation code table to find the entry that matches the statement operation code. The table, resident in BLOPLKUP, is described in Subsection 6.1.

ENTRY: This routine is entered at location BLOPLKUP from the new statement routine of BLPAS1 or BLPAS2. The address of the requested operation code is an in-line parameter in the calling sequence.

OPERATION: This routine performs a binary search of the operation code table. If no matching entry is found, the address of a dummy entry is returned to indicate that the search was unsuccessful.

EXIT: This routine exits to the calling program. The address of the matching table entry (or of a dummy entry) is returned in a general register.

2.4 SYMBOL TABLE SEARCH ROUTINE (BLSLKUP)

FUNCTION: This routine (Chart FA) searches the symbol table to find an unused location or the location of a specified symbol. The format of the symbol table is described in Subsection 6.1.

ENTRY: This routine is entered at BLSLKUP from a number of routines, each of which provides the address of the symbol sought and an error exit address.

OPERATION: This routine uses a 'hashing' algorithm to develop the search start address from the 8-character symbol and its 1-character qualifier. The search then proceeds sequentially until the first unused entry or a matching entry is found. If the end of the table is reached, the search restarts at the beginning of the table and moves through to the end. If the table is full and no match is found, the routine loads the address of a dummy entry into the output register to indicate that the symbol table is full.

EXIT: This routine has two exits, both in the calling routine. The normal exit is taken when the routine finds an unused location or a matching entry. If the search was fruitless, the routine places the address of a dummy entry in the output register and takes the error exit.

2.5 CONSTANT MODIFIER (BLCONMOD)

Modifier BLCONMOD validates the multiplicity (duplication factor), type, length, scaling, and value list of a literal or DC statement. Correctly defined elements are evaluated and then stored as binary values in an output table. (See Subsection 6.3 for the BLCONMOD output format.) Errors in any element result in a diagnostic code, the zeroing of all element fields in the output table, and a return to the calling program.

Modifier BLCONMOD consists of the BLCONMOD mainline and 14 logically distinct routines and subroutines, each clearly marked in the assembly listing. They are identified by their primary entry point. The names and functions of these routines and subroutines are as follows:

1. BLCONMOD Mainline: This controls BLCONMOD processing, using internal tables to verify modifiers and to call appropriate routines in correct sequence.

2. MMULTY Routine: This routine determines multiplicity when it is specified by the input statement.

3. MHOLD Routine: This routine assigns a multiplicity of 1 when no explicit value is specified.

4. TTYPE Routine: This routine determines the type of constant specified by the input statement.

5. THOLD Routine: This routine sets up for a C-type constant when no explicit type is given.

6. LLNGTH Routine: This routine determines the byte and bit length of a constant, using the explicit length given in the input statement.

7. LHOLD Routine: This routine develops the length, according to type, for a constant in which length is implied.

8. SSCALE Routine: This routine evaluates scaling and initiates value list handling.

9. CCI Routine: This routine counts the number of EBCDIC characters specified in the value list of a C-type constant.

10. HHI Routine: This routine counts the number of hexadecimal characters specified in the value list of an X-type constant.

11. VVI Routine: This routine checks the number of constants (i.e., commas) in the value list of an A-, S-, F-, H-, E-, or D-type constant.

12. ZZI Routine: This routine finds the total number of characters in the value list of P- and Z-type constants.

13. CERR Routine: This routine generates a diagnostic code, moves it to the output table, and sets up for program exit.

14. DSUBRT Subroutine: This subroutine calls BLEXVAL to evaluate an expression and then checks for errors.

15. INTGER Subroutine: This subroutine sets up an expression for evaluation by the DSUBRT subroutine and checks for errors.

2.5.1 BLCONMOD Mainline

FUNCTION: This part of BLCONMOD (Chart GA) controls processing by calling the proper routine to handle each element of a literal or DC statement. The mainline also performs initialization and termination functions for the entire program.

ENTRY: The mainline has one primary entry point, location BLCONMOD, which may be entered from the BLPAS1 and BLPAS2 programs. A secondary entry point, location FEXIT2, is entered by routines within BLCONMOD to terminate processing. (The input format is described in Subsection 6.3.)

OPERATION: Starting with the first character of the input statement, the mainline uses a combination of table lookups to decide which routine to call. A general internal table is used to translate a character from the input statement into a code number. This number is then translated into a routine address by one of four secondary tables. Each secondary table corresponds to a kind of element: multiplicity, type, length, or scaling factor. The multiplicity table is called first; the others are used in order as returns are made to the mainline.

If any element is found invalid during table lookups, mainline transfers control to the CERR routine to generate a diagnostic and prepare for exit.

EXIT: Final exit from mainline, and thus final exit from BLCONMOD, is to the address specified by the calling program.

Initially, based on its table lookup, the mainline can exit to any nine routines within BLCONMOND.

1. MMULTY routine, taken when multiplicity is specified in the input statement.

2. MHOLD routine, used when multiplicity is implicit.

3. TTYPE routine, taken when type is specified by the input statement.

4. THOLD routine, used for an implicit type.

5. LLNGTH routine, used when length is specified in the input statement.

6. LHOLD routine, used for an implicit length.

7. SSCALE routine, used when scaling is specified in the input statement.

8. SHOLD routine, used when scaling is implicit.

9. CERR routine, taken whenever an error is detected during table lookup.

2.5.2 MMULTY Routine

FUNCTION: This routine (Chart GB) determines the multiplicity (duplication factor) of a constant.

ENTRY: The routine is entered at location MMULTY from the BLCONMOD mainline when an input statement begins with a left parenthesis or a numeric character.

OPERATION: The routine calls the INTGER subroutine to evaluate the initial expression of the input statement. Upon return, which is made only if the expression is a valid multiplicity specification, the routine moves the multiplicity value to the BLCONMOD output table.

EXIT: Return is to the BLCONMOD mainline.

2.5.3 MHOLD Routine

FUNCTION: This routine (Chart GB) moves a multiplicity value of 1 to the BLCONMOD output table.

ENTRY: The routine is entered at location MHOLD from the BLCONMOD mainline when multiplicity is implicit.

EXIT: Return is to the BLCONMOD mainline.

2.5.4 INTGER Subroutine

FUNCTION: This subroutine (Chart GB) sets up an expression for evaluation and checks that the resulting value is valid.

ENTRY: There is one entry point, location INTGER. It may be entered from either the MMULTY or LLNGTH routine.

OPERATION: If the initial character of the expression is a left parenthesis, the subroutine immediately transfers to the DSUBRT subroutine for expression evaluation. Otherwise, it first isolates the expression by temporarily replacing the first non-numeric character with a blank, and then transfers to DSUBRT.

Upon return from DSUBRT, tests are made to ensure that the expression is valid before returning to the calling routine. If an error is detected, control passes to the CERR routine.

EXIT: This subroutine may exit to either of two locations.

1. To the calling routine. This is the normal exit.

2. To the CERR routine if an error is found. This, in effect, marks the end of BLCONMOD processing for the current statement.

COMMENT: In a normal exit, this subroutine sets up the next character for examination by the BLCONMOD mainline.

2.5.5 TTYPE Routine

FUNCTION: This routine (Chart GC) establishes the type code for an input statement by comparing the current input character with an internal table. The code is then moved to the BLCONMOD output table.

ENTRY: This routine is entered at location TTYPE from the BLCONMOD mainline when type is specified by the input statement.

EXIT: This routine returns to the BLCONMOD mainline.

2.5.6 THOLD Routine

FUNCTION: This routine (Chart GC) moves the type code for a C-type constant to the BLCONMOD output table.

ENTRY: The routine is entered at location THOLD from the BLCONMOD mainline when the input statement type is implicit.

EXIT: Return is to the BLCONMOD mainline.

2.5.7 LLNGTH Routine

FUNCTION: This routine (Chart GD) defines and checks the explicit length given in an input statement.

ENTRY: There is one entry point, location LLNGTH. It is entered from the BLCONMOD mainline when the input statement specified a length (signaled by the character L).

OPERATION: The routine begins by locating the start of the length expression, which must be either a number or a left parenthesis. If a decimal point is first met, meaning the expression refers to bits, the branch to location LBIT1 is taken. Otherwise the expression is assumed to indicate bytes, and the branch is to location LBYTE. There, after return from the INTGER subroutine, a second check is made to see if a bit expression has also been included in the length. If so, the routine looks back to evaluate the bit expression before proceeding.

After expression evaluation, the routine calculates and moves to the BLCONMOD output table the total number of bits and the total number of bytes (rounded, if needed, to the next higher byte boundary). Tests are then made to ensure that the length value agrees with the statement type. If the length proves valid, LLNGTH stores the ''TRUE'' length (byte length less 1) and an alignment code (1) before returning control to the BLCONMOD mainline.

EXIT: This routine may exit to either of two locations.

1. If the length is valid, return is to the BLCONMOD mainline.

2. If an error is detected, control passes to the CERR routine.

2.5.8 LHOLD Routine

FUNCTION: This routine (Chart GE) finds the bit and byte lengths and the alignment code for an input statement with implicit length.

ENTRY: The routine is entered at location LHOLD from the BLCONMOD mainline when an input statement does not specify length.

OPERATION: This routine uses internal tables and the statement's type code to develop the total byte and bit lengths and the alignment code for the constant. These values are moved to the BLCONMOD output table.

EXIT: Exit is to the calling point in the BLCONMOD mainline.

2.5.9 SSCALE Routine

FUNCTION: This routine (Chart GF) determines the internal scaling requested in a D-, E-, F-, or H-type constant. For all types of constants, it initiates value list processing.

ENTRY: This routine has two entry points.

1. Location SSCALE, entered from the BLCONMOD mainline when scaling is specified (indicated by an S) in the input statement.

2. Location SHOLD, entered from the BLCONMOD mainline when scaling is not specified.

OPERATION: When scaling is requested, the routine validates the scaling expression and sets up for evaluation by the DSUBRT Subroutine. Upon return, tests are made to ensure that the scaling value is valid before it is moved to the BLCONMOD output table. Any error results in an immediate exit to the CERR routine.

After the scaling process is finished or when scaling is not specified, the routine sets up for value list operations. The value list for an A- or S-type constant must start with a left parenthesis; the list for all other types must begin with a single quote. Any error found here results in a branch to an error sequence in the CCI routine to set up a diagnostic before passing control on to the CERR routine. For a proper value list, the routine exits according to the constant type.

EXIT: This routine may exit to any of seven locations.

1. Location CERR, in the CERR routine, taken when an error is discovered during scaling operations.

2. Location FEXIT2, in the BLCONMOD mainline, taken when there is no value list or when the value list indicates a literal. This leads to the normal exit from the BLCONMOD program.

3. Location CCBRN1, in the CCI routine, used when an error is detected during value list handling.

4. Location VVI, in the VVI routine, taken for an A- or S-type (address constant), a D- or E-type (floating-point constant), or an F- or H-type (fixed-point constant) value list.

5. Location CCI, in the CCI routine, used for a C-type (character constant) value list.

6. Location HHI, in the HHI routine, taken for an X-type (hexadecimal constant) value list.

7. Location ZZI, in the ZZI routine, taken for a P- or Z-type (decimal constant) value list.

ERRORS: Any error is communicated to the CERR routine which causes the following message to be printed.

ERROR IN VALUE LIST

2.5.10 CCI Routine

FUNCTION: This routine (Chart GG) counts the number of characters in the value list of a C-type (character) constant.

ENTRY: This routine has three entry points.

1. Location CCI, the primary entry point, entered from the SSCALE routine.

2. Location CCBRN1, entered from the SSCALE, HHI, VVI, and ZZI routines when an error is detected in the value list.

3. Location CCMPD, entered from the HHI routine when that routine has finished processing.

OPERATION: The routine counts all characters in the value list until it finds the quote that marks the end of the list. If the end of the statement is reached without finding the quote, an error exit is taken.

After the quote is found, the next operation depends on whether a length was specified. If so, the routine returns control directly to the BLCONMOD mainline to conclude BLCONMOD operations. For an implicit length, the character count is used to develop the byte and bit lengths for the constant. These values are stored in the BLCONMOD output table before exit.

EXIT: This routine may exit to either of two locations.

1. Location FEXIT2, in the BLCONMOD mainline, taken at the end of normal processing. This leads to the exit from the BLCONMOD program.

2. Location ISDIX, in the CERR routine, used if an error was detected.

ERRORS: This routine sets up a diagnostic code for the CERR routine when an error is discovered in the value list. As a result, the following message will appear in the assembly listing.

ERROR IN VALUE LIST

2.5.11 HHI Routine

FUNCTION: This routine (Chart GH) counts the number of characters in the value list of an X-type (hexadecimal) constant.

ENTRY: The routine has one entry point, location HHI, which is entered from the SSCALE routine.

OPERATION: The routine counts all characters until it finds the quote ending the value list. If this quote is not found before the statement end, an error exit is taken.

If the constant has a specified length, the routine then transfers control to set up a normal exit from the BLCONMOD program. For a constant with implicit length, it uses the character count to generate byte and bit lengths for the output table before relinquishing control.

EXIT: This routine may exit to either of two locations.

1. Location CCMPD, the CCI routine, taken at the end of normal processing. This saves the address of the end of the value list before passing control to the BLCONMOD mainline for program exit.

2. Location CCBRN1, in the CCI routine, used if an error is found. This sets up a specific diagnostic code before transferring control the CERR routine.

ERRORS: If an error is discovered, a diagnostic code is set up for the following message.

ERROR IN VALUE LIST

2.5.12 VVI Routine

FUNCTION: This routine (Chart GI) counts the number of constants in the value list of A- and S-type (address), D- and E-type (floating-point), and F- and H-type (fixed-point) constants.

ENTRY: There is one entry point, location VVI, which is entered from the SSCALE routine.

OPERATION: To start, this routine loads the current bit length value (available in the BLCONMOD output table) into a counter and sets the value list terminator to agree with the type of constant. A character-by-character scan is then made until the terminator appears; expressions are bypassed. As each comma is detected, marking the end of a constant, the initial bit length value is added to the counter value.

When the terminator occurs, the routine rounds the bit value in the counter to the next higher byte and converts it to bytes. This new byte length is then moved to the BLCONMOD output table.

If a left parenthesis is encountered during the scan loop, the routine calls BLEXVAL to find the expression terminator, and processing then continues with the next character after the expression terminator. In this way, VVI skips any parentheses or quotes within a constant. An error here, or encountering the end of the value list before the terminator, results in a branch to the CCI routine to set up a diagnostic.

EXIT: This routine can exit to either of two locations.

1. Location FEXIT2, in the BLCONMOD mainline, taken at the end of normal processing.

2. Location CCBRN1, in the CCI routine, used when an error is detected.

ERRORS: The following diagnostic message is initiated when an error is found.

ERROR IN VALUE LIST

2.5.13 ZZI Routine

FUNCTION: This routine (Chart GJ) counts the number of characters in the value list of a P- or Z-type (decimal) constant. For a constant with implicit length, it uses count to develop a length value.

ENTRY: Location ZZI, the only entry point to this routine, is entered from the SSCALE routine.

OPERATION: If a length was specified for the constant, the routine merely locates the value-list terminator (a quote followed by a blank) and exits.

For an implicit-length constant, the routine scans the value list character-by-character until a decimal point or terminator appears. The number of characters before the decimal point, not including the sign, is saved in one counter. When the decimal point is found, the routine switches to the loop at location ZMOD2 and continues the scan for the terminator. A second counter now records the number of characters to the right of the decimal point.

Once the terminator appears, the routine calculates the constant length by adding together the two counter values, shifting accordingly if the value list is packed, and getting the total byte and bit counts. The total counts and the length are then moved to the BLCONMOD output table before exit.

EXIT: This routine can exit to either of two locations.

1. Location FEXIT2, in the BLCONMOD mainline, taken at the end of normal processing.

2. Location CCBRN1, in the CCI routine, used if the value list terminator is not found before the end of input.

ERRORS: The routine initiates the following diagnostic message when an error is detected.

ERROR IN VALUE LIST

2.5.14 DSUBRT Subroutine

FUNCTION: This subroutine (Chart GK) calls the BLEXVAL program to evaluate an expression and, upon return, checks the BLEXVAL output for errors.

ENTRY: There is one entry point, location DSUBRT; it is entered from the INTGER subroutine and the SSCALE routine.

EXIT: The subroutine exits to either of two locations.

1. If no error is found, exit is to the calling routine.

2. If BLEXVAL has indicated an error or if the expression proves relocatable, the subroutine transfers control to the CERR routine.

2.5.15 CERR Routine

FUNCTION: This routine (Chart GK) controls error handling for the BLCONMOD program.

ENTRY: The routine has two entry points.

1. Location CERR is entered from many places in BLCONMOD when any error, except one in a value list, is discovered.

2. Location ISDIX is entered from the CCI routine when a value list error is found. All routines involved with the value list (SSCALE, CCI, VVI, HHI, and ZZI) make use of this entry via the CCI routine.

OPERATION: When entered at CERR, the routine generates an appropriate diagnostic code by checking which routine-finding table was last used by the BLCONMOD mainline. Any entry at ISDIX provides a value list error code in the call.

The routine next moves the diagnostic code to the BLCONMOD output table along with an error count, then zeros all other fields in the table before passing control to the mainline.

EXIT: Exit is to location FEXIT2 in the BLCONMOD mainline, which results in a return to the calling program. In effect, BLCONMOD terminates operations as soon as an error appears.

ERRORS: This routine, and therefore the BLCONMOD program, can initiate two error messages. They are:

ERROR IN MODIFIER(S)
ERROR IN VALUE LIST

2.6 EVALUATE EXPRESSION ROUTINE (BLEXVAL)

The BLEXVAL routine consists of three logical sections: the BLEXVAL mainline itself and two subroutines contained within the BLEXVAL coding. These sections, clearly marked on the assembly listing, are described separately on the following pages. Their names and functions are as follows.

1. BLEXVAL Routine. The mainline of BLEXVAL, this routine evaluates an expression, making calls to the SCANX subroutine as necessary.

2. SCANX Subroutine: This section of BLEXVAL evaluates an element of an expression, calling the SGET subroutine for symbol information and returning control to the BLEXVAL routine.

3. SGET Subroutine. This subroutine finds the location of an entry in the symbol table or system symbol table that matches a specified symbol. It normally returns control to the SCANX subroutine.

2.6.1 BLEXVAL Routine

FUNCTION: This routine (Charts HA, HB, HC) examines an expression to determine its value (i.e., storage location), length, scale factor, relocation ID, and type. It also validates the expression and issues an appropriate diagnostic code if an error is found. The results of the evaluation are stored in an output table for the use of the calling routine.

ENTRY: This routine is entered at location BLEXVAL from the BLPAS1, BLCONMOD, BLCONVAL, and BLPAS2 programs. The calling program provides, in a general register, the address of the expression to be evaluated.

OPERATION: The routine begins by calling the BLBRKUP routine to divide an expression into elements, each of which is returned left-adjusted in a word or words. Routine BLBRKUP also returns a count list that indicates the number of nonblank characters in each element.

The routine next calls SCANX to evaluate the first element. The results of that evaluation define the attributes of that entire expression, and are stored in the BLEXVAL output table. (See Subsection 6.3 for a detailed description of the output format.) Additional calls are made to the SCANX subroutine for each remaining element in order to collect data to compute the value and relocation ID of the expression. The results of these elements are stored temporarily in two internal tables.

The loop beginning at location RLOOP checks for an invalid complexly-relocatable expression and develops the relocation ID for the expression. If the relocation ID of an element is nonzero, the loop checks whether the ID is involved in a multiplication or division (an error). The current ID is then compared with the relocation ID of the preceding element and, if they are the same, a counter is either increased or decreased by 1 depending on whether the sign of the ID is positive or negative. When all location IDs in the expression have been tested, this counter value indicates whether the expression is absolute, simply relocatable, or negatively relocatable. Any other value in the counter indicates an invalid complex expression.

The two loops (starting at locations VALOOP and VS2LP) develop the value of the expression, according to the expression operators. In the first loop, element values are multiplied and divided. In the second, additions and subtractions are performed to develop the final value of the expression.

The remainder of this routine is concerned with issuing diagnostic codes, indicating error type, and developing the error count. The diagnostic code and the corresponding invalid element are repeated in the BLEXVAL output table for every error encountered.

EXIT: This routine exits to the calling program. The address of the BLEXVAL output table and of the expression terminator are returned in general registers.

ERRORS: This routine sets up four pieces of error information in its output table: error type, error count, diagnostic code, and error symbol. The table also includes any diagnostic information generated by the SCANX or SGET subroutines.

Error type is shown by the following bit settings of an otherwise-unused whole word:

Bit Set	Error Type
31	Expression cannot be evaluated; it is complexly-relocatable
30	Expression may be in error (for an expression containing no elements)
29	Rest of statement cannot be evaulated; improper symbol usage in current expression
28	Expression is only partially evaluated because of truncation
27	Multi-defined symbol appeared

Error count is the number of diagnostics issued for the current expression.

The following diagnostic codes are issued by the BLEXVAL routine itself (see descriptions of the SCANX and SGET subroutines for other codes that may appear in the BLEXVAL output table):

Code	Message
1	FIELD n HAS INVALID PUNCTUATION*
2	FIELD n HAS AN INVALID CHARACTER*
3	FIELD n HAS A SYMBOL OR NUMBER*
4	FIELD n HAS AN EXPRESSION WHICH IS TOO LONG OR COMPLEX*
7	FIELD n IS INVALIDLY COMPLEX
8	FIELD n HAS A VOID FIELD
10	FIELD n HAS A RELOCATABLE SYMBOL WHICH IS MULTIPLIED OR DIVIDED
11	FIELD n HAS TOO MANY ELEMENTS IN AN EXPRESSION
44	FIELD n HAS DIVISION WHICH RESULTED IN ZERO QUOTIENT
48	FIELD n HAS TWO CONSECUTIVE QUOTES AFTER SYMBOL X

*These codes are taken directly from the BLBRKUP return.

The error symbol, set up by the SGET subroutine, is issued immediately after a diagnostic for an undefined or multi-defined symbol and will later be printed in the assembly listing along with the diagnostic message. If no such symbol exists, the two-word area is set to the current operand number. This was initialized before entry into BLEXVAL.

2.6.2 SCANX Subroutine

FUNCTION: This subroutine (Charts HD, HE) examines an element of an expression. Depending on the individual element, SCANX converts a decimal number to binary; identifies an operator or a special element such as a self-defining value; sets up value and relocation-ID information; and, from the first element of an expression, establishes expression attributes.

ENTRY: The subroutine is entered at location RSCAN1 or RSCAN2 from the BLEXVAL routine. The RSCAN1 entry is used for the first element of an expression; all other elements enter at RSCAN2. The only difference between the two entries is in the setting of a switch to handle expression attributes.

OPERATION: The subroutine begins with a test of the count generated by the BLBRKUP routine for the element. If only one character appears, a check is made for an operator (an *, /, +, -, or '); the relocation sign or operator is set accordingly; and the next element field is examined. An extra check is made if an asterisk is found in order to distinguish between its use as a multiplier and as a location counter reference. A quote is handled by part of the two-character checking sequence.

The subroutine examines a two-character element for a hexadecimal or character self-defining value (X' or C') or for a symbolic definition of length, type, or scale factor (L', T', or S', followed by a symbol). For the latter, information is fetched from the symbol table through a call to the SGET subroutine. A character self-defining value is stored exactly as received. The hexadecimal, however, requires conversion to binary, which is done by dropping the zone portion of each character and adding 9 for hexadecimal digits A through F.

All nonspecial characters and all elements with three or more characters are processed either as numbers or as a symbol. The subroutine multiplies an integer by powers of 10 to convert it to binary (e.g., 264 = 2x100+6x10+4x1), then stores the result for BLEXVAL. The SGET subroutine is called to handle symbols, and the information it returns is used to collect length, type, scale factor, relocation, value, and control information for the first element of an expression. Only relocation and value information is saved for symbols found in the remaining elements.

EXIT: This subroutine has four exits, all to the BLEXVAL routine: a normal return; an expression terminator return; and two error returns. One error is taken for an invalid two-character element ending with a quote; the other is taken if SGET has found a multi-defined symbol.

ERRORS: The only diagnostic code set up by this subroutine is:

6 FIELD n HAS INVALID USE OF*

This code appears in the BLEXVAL output table.

2.6.3 SGET Subroutine

FUNCTION: This subroutine (Chart HF) looks up a symbol in the symbol table or system symbol table and saves the corresponding entry address. It also generates diagnostic information for undefined and multi-defined symbols.

ENTRY: This subroutine is entered at location SGET from the SCANX subroutine.

OPERATION: After establishing the qualifier for the symbol, the subroutine calls the BLSLKUP (symbol lookup) routine to find the address of the symbol's entry in the symbol table. If a matching entry is not found, SGET then searches the system symbol table. Diagnostic messages are initiated if the symbol remains undefined or if BLSLKUP finds it to be multi-defined.

EXIT: Depending on the results of the search, SGET returns either to the SCANX subroutine or to the BLEXVAL routine. The subroutine returns to SCANX when the symbol entry is found or when the symbol is multi-defined. In the latter case, SGET sets an indicator within SCANX so that diagnostic information is passed on to BLEXVAL. When the symbol is undefined, the subroutine exits directly to the BLEXVAL routine.

ERORRS: The subroutine moves an undefined or multi-defined symbol to an error list for subsequent printing along with the diagnostic message, and sets up these diagnostic codes for BLEXVAL.

Code	Meaning
5	Undefined symbol.
12	Multi-defined symbol.
46	Symbol has not been previously defined; i.e., an undefined symbol has been detected during pass 1.
78	Symbol not defined, perhaps because of symbol table overflow. This is issued if a symbol table overflow has occurred and the symbol cannot be found.

2.7 BREAK UP EXPRESSION ROUTINE (BLBRKUP)

FUNCTION: This routine (Chart JA) separates an expression into its logical elements and makes a syntactic check of each element. If an error is detected, the routine sets up a diagnostic code to inform the calling routine.

ENTRY: The routine is entered at location BLBRKUP from BLPAS1, BLPAS2, and BLEXVAL and from the symbolic analyzer. The calling routine supplies in a general register the address of the expression to be processed.

OPERATION: The routine forms a general loop to process an expression character-by-character, using two internal tables to decide the specific path taken for each character. The tables also designate the character's syntactic type so that the two BLBRKUP subroutines (STORE and SKIP) can check for syntax errors. These subroutines are described on the following pages.

The routine builds two output tables: the output list and the count table. The output list contains each element or operator left-adjusted in a word or, for an element with more than four characters, in two words. For example, the expression L'BOX+AB.3 would appear in the following form:

```
|L'bb | BOXb | +bbb | ABbb | .3bb|
           (b = blank)
```

The count table presents the number of characters in each element, right-justified in a word. The first word of the table includes any diagnostic code that applies to the expression. For the sample above, the count table would appear as follows.

```
| bbb0 | bbb2 | bbb3 | bbb1 | bbb2 |
                     | bbb2 | bbb0 |
```

The zero in the last word indicates the expression terminator, which, in this case, is a blank. The zero in the first word means no error was detected.

Since the operation of some BLBRKUP coding segments is not easily seen because of the internal table lookup, these segments are described below.

The segment beginning at OP6 is called only when a double quote has been found. One of these is saved and used; the other is skipped by means of this segment.

The OP7 segment is called when table lookup finds an expression with invalid punctuation; a diagnostic code is set and the routine exits.

The OP8 segment is used for a character self-defined value, indicated by C' or C@. The segment sets up an internal table so that it accepts only the type of quote (' or @) that appeared at the start of the value. In this way, BLBRKUP can distinguish the terminating quote. The OP9 segment is called when the terminating quote is found, and reinitializes the internal table to accept either type of quote.

EXIT: This routine returns to the calling routine. The routine loads into three general registers the address of the output list, the count table, and the expression terminator as described in Subsection 6.3.

ERRORS: Four diagnostic codes can be set.

1. Invalid or missing punctuation

2. Invalid character

3. Symbol or number too long or too short

4. Expression too long or complex

The code is returned to the calling routine in the first word of the count table. The calling routine will then use the code to set up the actual error message in the assembly listing.

2.7.1 STORE Subroutine

FUNCTION: This subroutine (Chart JB), operating within the BLBRKUP routine, moves a character to the output list and increases counters to prepare for the next character. Using information given by the syntax table, the subroutine also checks for a number of symbols longer than 8 characters.

ENTRY: Entry is a location STORE from the BLBRKUP routine.

EXIT: The subroutine returns to the calling point in the BLBRKUP routine. If a length error is found, the subroutine exits to location OP3.

ERRORS: If the subroutine detects an invalid length, it sets up diagnostic 3 (symbol or number too long) for BLBRKUP before exiting to location OP3.

2.7.2 SKIP Subroutine

FUNCTION: This subroutine (Chart JB), operating within the BLBRKUP routine, steps to the next word in the output list and sets up for a new element. It also inserts blanks, if necessary, to fill out the preceding word and checks for an invalid expression.

ENTRY: Entry is at location SKIP from the BLBRKUP routine.

EXIT: The subroutine always returns to the calling point in the BLBRKUP routine, normally to the address given in a general register. If an error is found, the subroutine exits to location OP3.

ERRORS: If the subroutine finds the expression too long or complex, it sets up diagnostic code 4 for BLBRKUP before exiting to location OP3.

2.8 BLPUNC2 ROUTINE

FUNCTION: This routine (Chart UA) punches TXT, ESD, RLD, DBG, LIB, and END cards, and links to the PAKTAP (or PAKTXT) subroutine to block the object code on the AUXIL tape.

ENTRY: The entry point for this routine is location BLPUNC2. It is entered from the PASS1 for ESD and LIB cards; from the PRINT routine for TXT cards, and from PASS2 for RLD, DBG, and END cards.

OPERATION: This routine processes one card each time it is called.

If punching was requested by the programmer, the routine punches a card through a call to the SYSPUN system I/O routine, including on it the identification number indicated by a counter. In the initial call, it also issues a $OBJ card. The punch operation ends when the counter value has been incremented, converted to decimal, and packed for use as the next card identification number.

If the current job is to be executed, the routine links to the PAKTAP subroutine (and through it to the PAKTXT subroutine) to put a card image or packed TXT record on .AUXIL. Return from PAKTAP or PAKTXT is to the instruction following the call. When return is from PAKTXT, the next PUNC2 call is to the alternate entry within PAKTXT.

EXIT: This routine exits to the calling point.

2.8.1 PAKTAP Subroutine

FUNCTION: Each time it is called, this subroutine (Chart UB) links to the PAKTXT subroutine and, when PAKTXT returns control, writes a card image or packed TXT record on .AUXIL.

ENTRY: This subroutine is entered at PAKTAP from the BLPUNC2 routine.

OPERATION: The subroutine begins by setting an indicator if the input is an END card; it then calls PAKTXT. No other operation is performed unless PAKTXT returns to this subroutine with a record ready for .AUXIL.

Upon return from PAKTXT, the subroutine uses a call to SYSWRS to place a record on .AUXIL. If this record was an END card, the subroutine returns to BLPUNC2 without selecting PAKTXT for the next pass through BLPUNC2, since no further writing is expected. If an END card is read as input but has not yet been placed on .AUXIL, the subroutine turns off the ENDSW indicator and recalls PAKTXT to write the record before final return to BLPUNC2.

EXIT: This subroutine exits to the calling point in the BLPUNC2 routine.

2.8.2 PAKTXT Subroutine

FUNCTION: This subroutine (Chart UC) packs TXT cards into a buffer until it finds a non-TXT card, a noncountiguous TXT card, or a buffer overflow. It then returns to the PAKTAP subroutine to place the packed TXT card record on .AUXIL. All other types of cards, including TXT cards for common storage, are presented to PAKTAP without packing.

ENTRY: This routine is entered at location PAKTXT from PAKTAP or from BLPUNC2.

OPERATION: When first entered, the subroutine examines the card image to determine if it is a non-common TXT card, the only type of card that is packed. If so, the subroutine turns on a TXT-started indicator, moves the card to the buffer, and establishes PAKTXT as the next entry (for BLPUNC2) before exiting to BLPUNC2. As subsequent TXT cards are read, their text portion is also moved to the buffer. This continues until a non-TXT card, a non-contiguous TXT card, or a buffer overflow appears.

When the current buffer load is terminated, the subroutine (OUTBUF) sets up address and length parameters and transfers to PAKTAP to put the buffer contents on .AUXIL. The card that terminates filling of the buffer is retained in a storage area. Since the next entry is through PAKTXT, with the TXT-started indicator on, the retained card is then moved into the buffer either as the start of a new TXT record or as the next output to PAKTAP if it is a common-TXT or non-TXT card.

Any card found before the first non-common TXT card is moved directly to the buffer (at EXITA) and immediately presented to PAKTAP. This operation handles the BLPAS1 output of ESD and LIB cards.

EXIT: The subroutine has two exits. One is to the PAKTAP subroutine for all writing operations; the other is to the PUN2 routine when writing is not desired.

2.9 BLLIST (LISTING GENERATOR) ROUTINE

FUNCTION: This routine (Chart VA) handles all printing to SYSOUT. It includes all logic to handle spacing and page ejection, either requested or due to line-count overflow.

ENTRY: The entry point for this routine is location BLLIST. It is entered from all portions of the assembler which produce printed output.

EXIT: This routine returns to the calling point.

Figures 2-1 and 2-2 illustrate the general logic flow of pass 1 and are followed by detailed flowcharts of the programs that comprise pass 1.

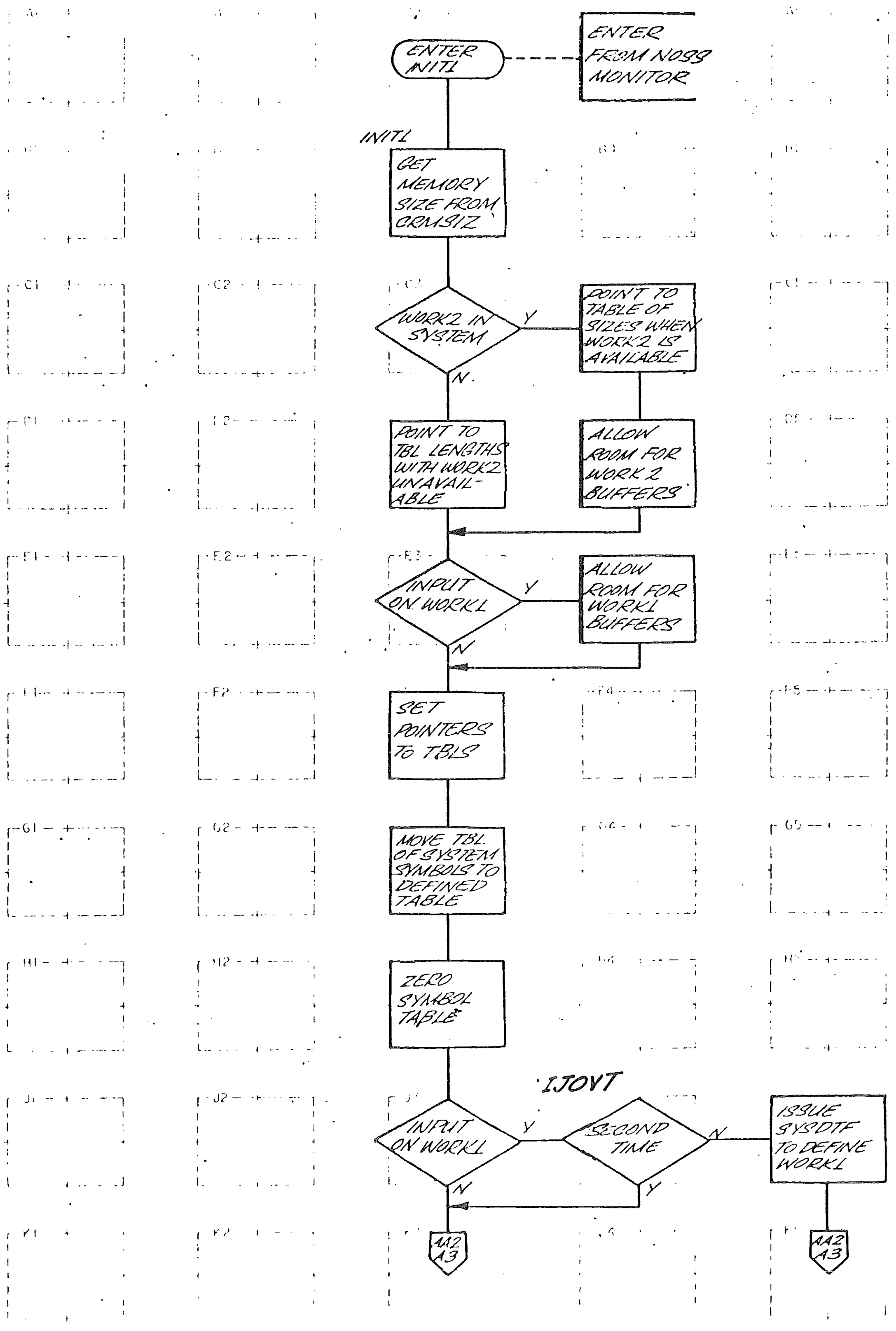

ENTER INITL
ENTER FROM NOSS MONITOR
INITL
GET MEMORY SIZE FROM CRMSIZ
WORK2 IN SYSTEM
Y
N
POINT TO TABLE OF SIZES WHEN WORK2 IS AVAILABLE
POINT TO TBL LENGTHS WITH WORK2 UNAVAIL-ABLE
ALLOW ROOM FOR WORK 2 BUFFERS
INPUT ON WORK1
Y
N
ALLOW ROOM FOR WORK1 BUFFERS
SET POINTERS TO TBLS
MOVE TBL OF SYSTEM SYMBOLS TO DEFINED TABLE
ZERO SYMBOL TABLE
IJOVT
INPUT ON WORK1
Y
N
SECOND TIME
N
Y
ISSUE SYSDTF TO DEFINE WORK1
AA2 A3
AA2 A3

CHART AA1. BAL, BAL INITIALIZATION AND SEQUENCE CONTROL (SHEET 1 OF 3)

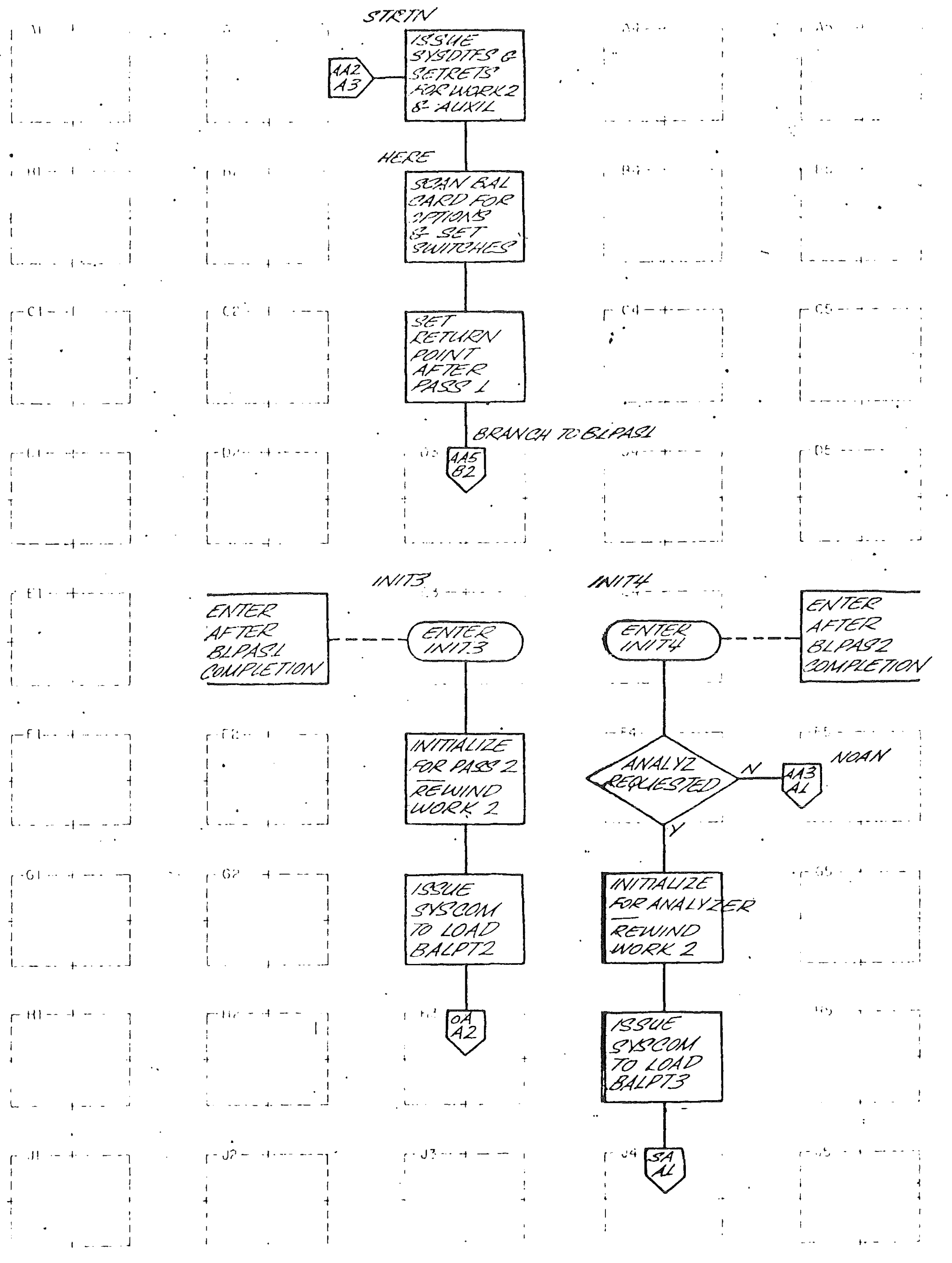
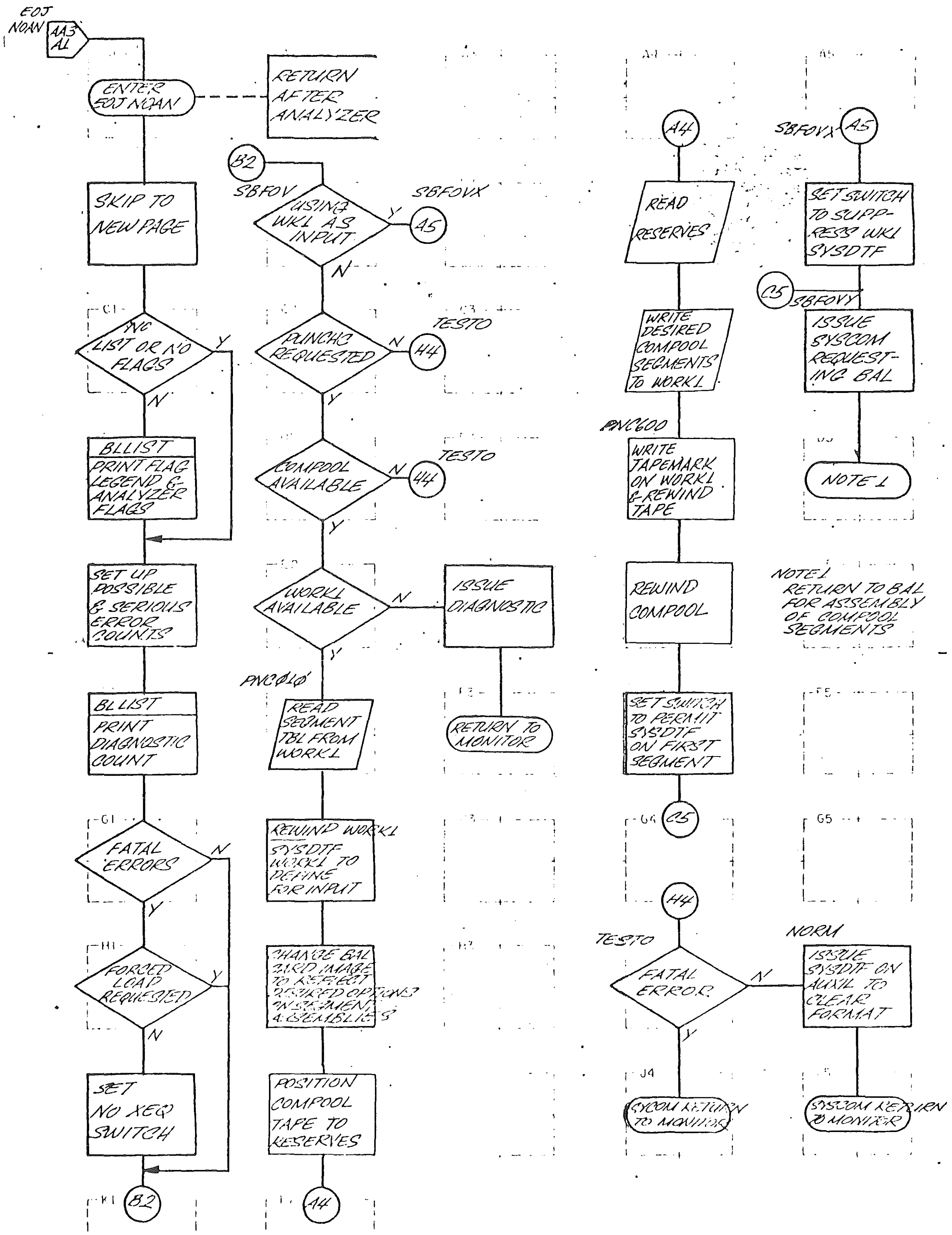

STRTN
AA2 A3
ISSUE SYSDTFS & SETRETS' FOR WORK2 & AUXIL
HERE
SCAN BAL CARD FOR OPTIONS & SET SWITCHES
SET RETURN POINT AFTER PASS 1
BRANCH TO BLPAS1
AA5 B2
INIT3
ENTER AFTER BLPAS1 COMPLETION
ENTER INIT3
INIT4
ENTER INIT4
ENTER AFTER BLPAS2 COMPLETION
INITIALIZE FOR PASS 2 REWIND WORK 2
ANALYZ REQUESTED
N
Y
AA3 A1
NOAN
ISSUE SYSCOM TO LOAD BALPT2
INITIALIZE FOR ANALYZER REWIND WORK 2
OA A2
ISSUE SYSCOM TO LOAD BALPT3
SA A1

CHART AA2. BAL, BAL INITIALIZATION AND SEQUENCE CONTROL (SHEET 2 OF 3)



EOJ NOAN
AA3 A1
ENTER EOJ NOAN
RETURN AFTER ANALYZER
SKIP TO NEW PAGE
NO LIST OR NO FLAGS
Y
N
BLLIST PRINT FLAG LEGEND & ANALYZER FLAGS
SET UP POSSIBLE & SERIOUS ERROR COUNTS
BLLIST PRINT DIAGNOSTIC COUNT
FATAL ERRORS
SET NO XEQ SWITCH
FORCED LOAD REQUESTED
B2
SBFOV
USING WK1 AS INPUT
SBFOVX
A5
PUNCHC REQUESTED
TESTO
A4
COMPOOL AVAILABLE
WORK1 AVAILABLE
ISSUE DIAGNOSTIC
RETURN TO MONITOR
PNC010
READ SEGMENT TBL FROM WORK1
REWIND WORK1 SYSDTF WORK1 TO DEFINE FOR INPUT
CHANGE BAL CARD IMAGE TO REFLECT DESIRED OPTIONS ON SEGMENT ASSEMBLIES
POSITION COMPOOL TAPE TO RESERVES
READ RESERVES
WRITE DESIRED COMPOOL SEGMENTS TO WORK1
PNC600
WRITE TAPEMARK ON WORK1 & REWIND TAPE
REWIND COMPOOL
SET SWITCH TO PERMIT SYSDTF ON FIRST SEGMENT
C5
H4
FATAL ERROR
SYSCOM RETURN TO MONITOR
SET SWITCH TO SUPPRESS WK1 SYSDTF
SBFOVY
ISSUE SYSCOM REQUESTING BAL
NOTE 1
NOTE 1 RETURN TO BAL FOR ASSEMBLY OF COMPOOL SEGMENTS
NORM
ISSUE SYSDTF ON AUXIL TO CLEAR FORMAT

CHART AA3. BAL, BAL INITIALIZATION AND SEQUENCE CONTROL (SHEET 3 OF 3)

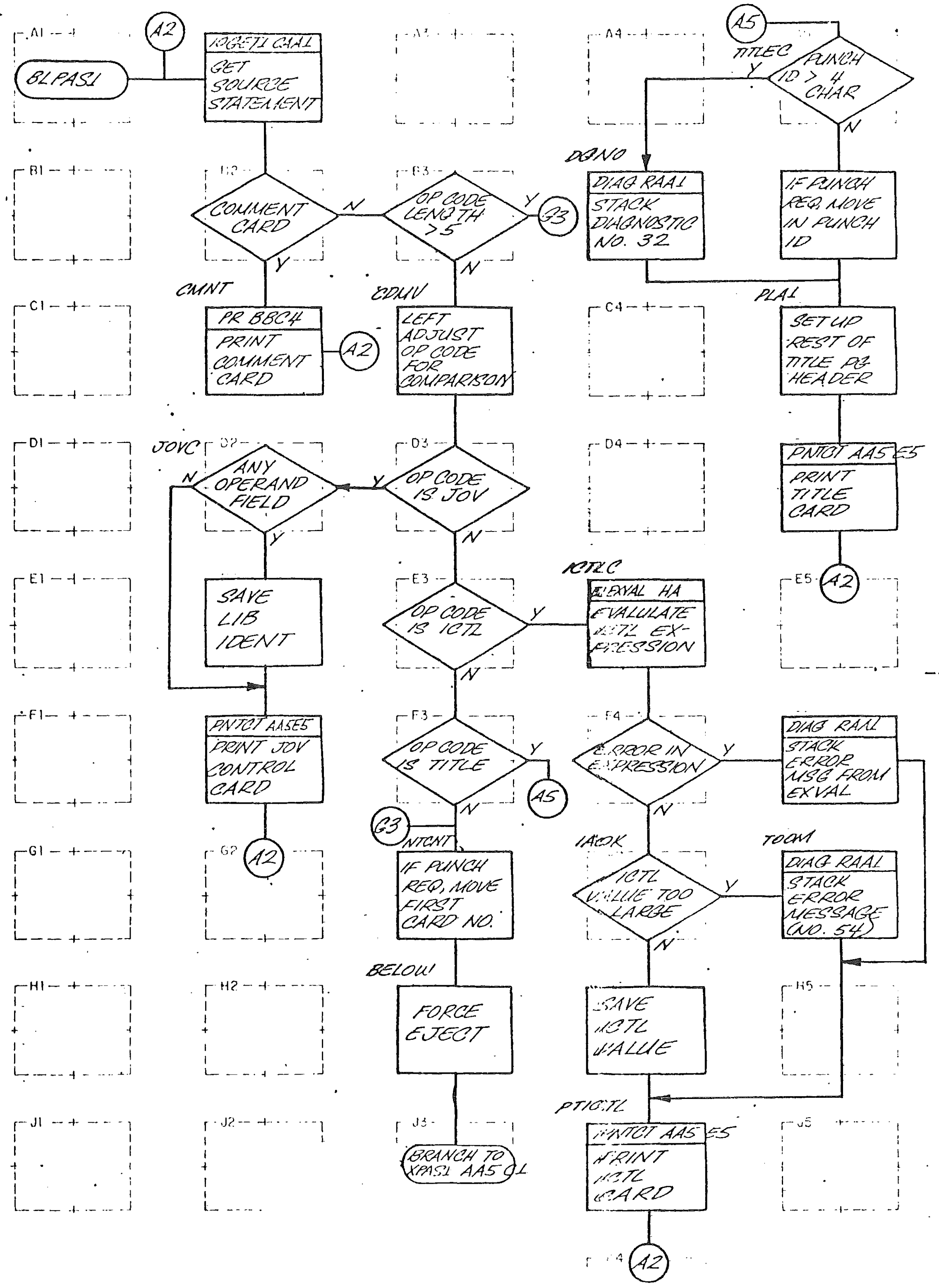
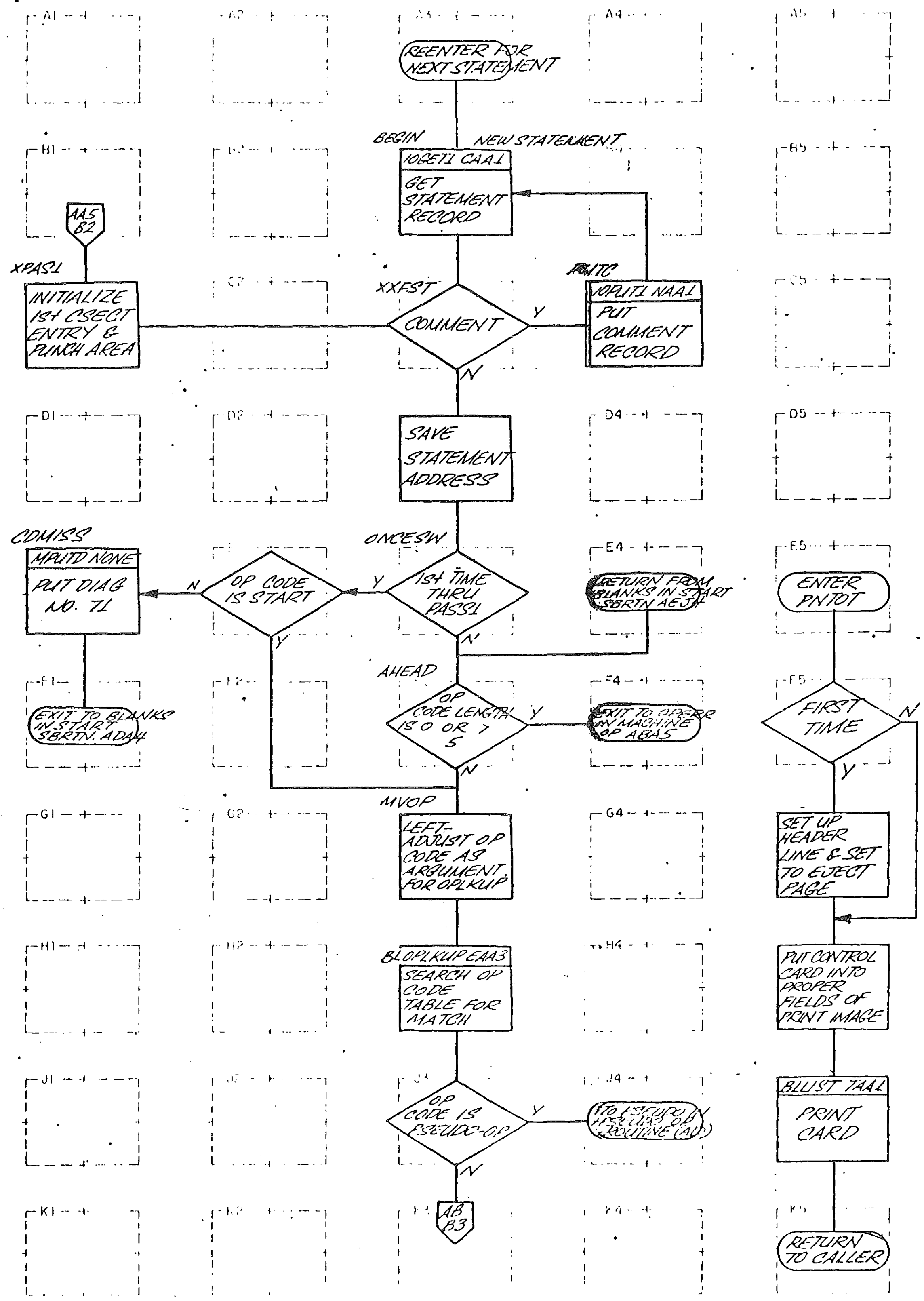
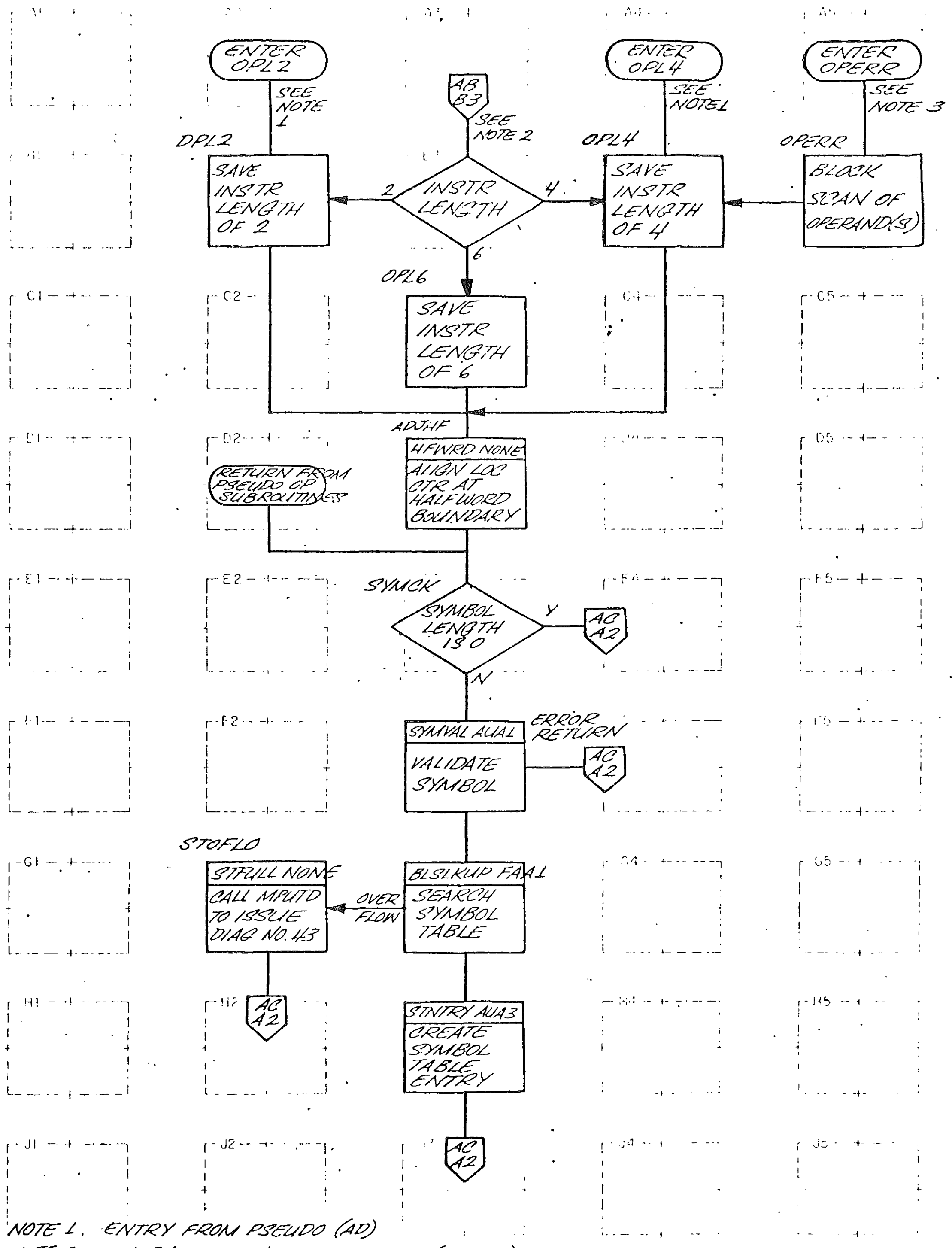


CHART AA4. BLPAS1 INITIALIZATION



REENTER FOR NEXT STATEMENT
BEGIN
NEW STATEMENT
10GET1 CAA1
GET STATEMENT RECORD
AA5 B2
XPAS1
INITIALIZE 1st CSECT ENTRY & PUNCH AREA
XXFST
COMMENT
Y
N
PNTC
10PUT1 NAA1
PUT COMMENT RECORD
SAVE STATEMENT ADDRESS
COMISS
MPUTD NONE
PUT DIAG NO. 71
N
OP CODE IS START
Y
ONCESW
1st TIME THRU PASS1
N
RETURN FROM BLANKS IN START SBRTN AEJ4
ENTER PNTOT
EXIT TO BLANKS IN START SBRTN. ADA4
AHEAD
OP CODE LENGTH IS 0 OR 7 5
Y
EXIT TO OPERR IN MACHINE OP ABA5
N
FIRST TIME
N
Y
MVOP
LEFT-ADJUST OP CODE AS ARGUMENT FOR OPLKUP
SET UP HEADER LINE & SET TO EJECT PAGE
BLOPLKUP EAA3
SEARCH OP CODE TABLE FOR MATCH
PUT CONTROL CARD INTO PROPER FIELDS OF PRINT IMAGE
OP CODE IS PSEUDO-OP
Y
TO PSEUDO OP ROUTINE (AD)
BLUST TAA1
PRINT CARD
N
AB B3
RETURN TO CALLER

CHART AA5. BLPAS1, BEGIN ROUTINES, PNTCT SUBROUTINE



NOTE 1. ENTRY FROM PSEUDO (AD)
NOTE 2. ENTRY FROM NEW STATEMENT (AA5,K3)
NOTE 3. ENTRY FROM NEW STMNT (AA5, F4) OR PSEUDO (AD)

CHART AB. MACHINE OPERATION ROUTINE (SHEET 1 OF 2)

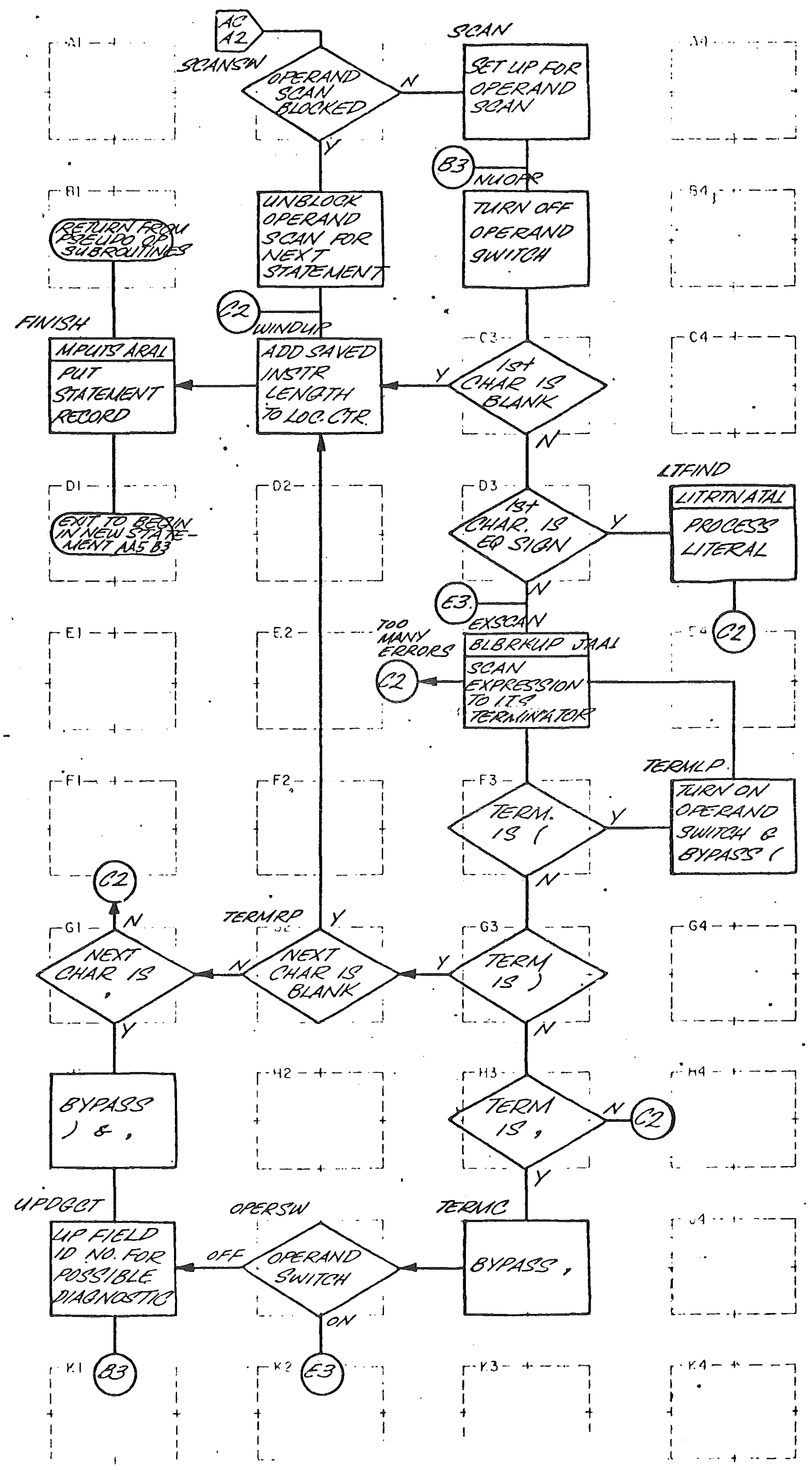


CHART AC. MACHINE OPERATION ROUTINE (SHEET 2 OF 2)

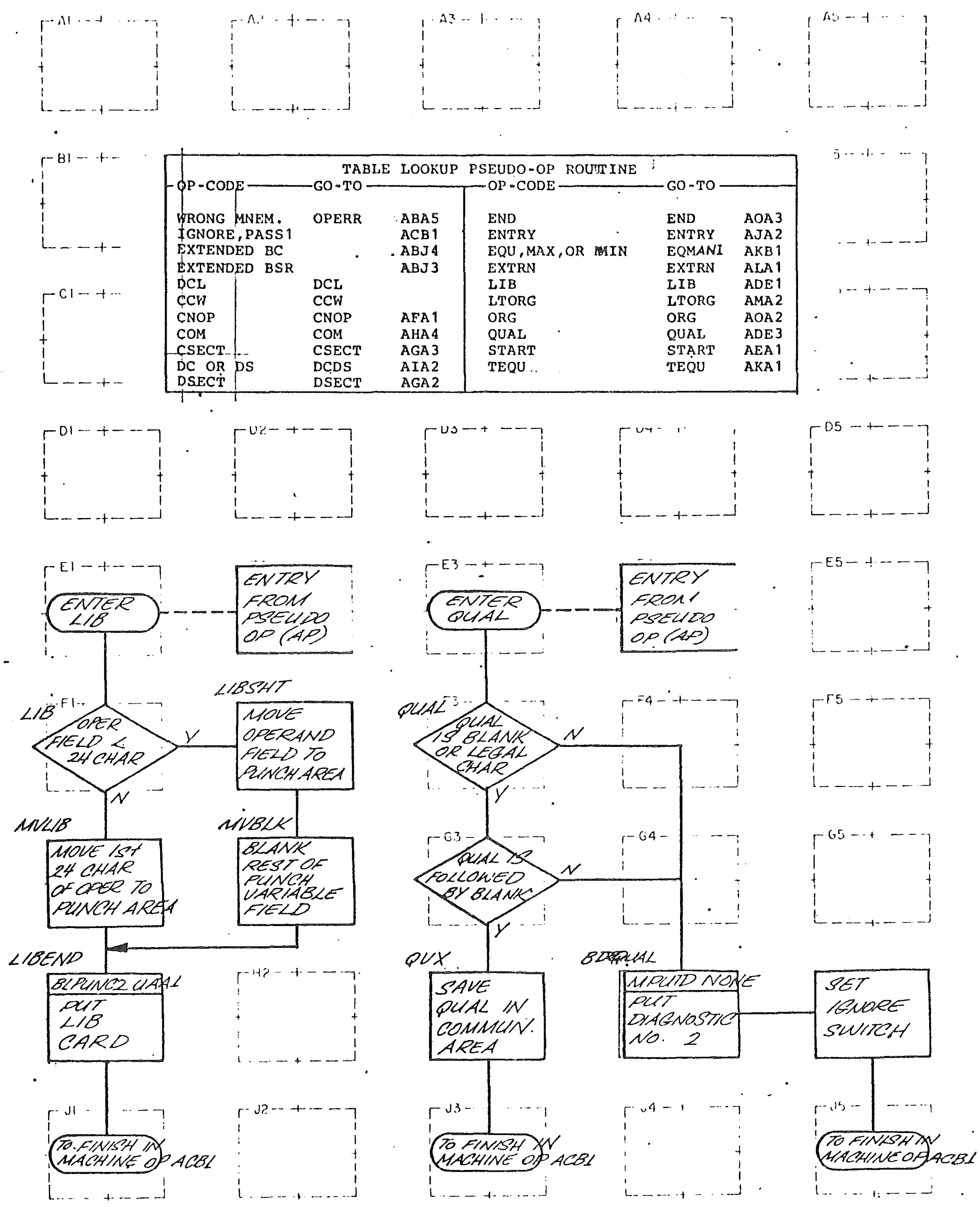

ENTER LIB
ENTRY FROM PSEUDO OP (AP)
ENTER QUAL
ENTRY FROM PSEUDO OP (AP)
LIB
OPER FIELD ≤ 24 CHAR
Y
N
LIBSHT
MOVE OPERAND FIELD TO PUNCH AREA
QUAL
QUAL IS BLANK OR LEGAL CHAR
N
Y
MVLIB
MOVE 1st 24 CHAR OF OPER TO PUNCH AREA
MVBLK
BLANK REST OF PUNCH VARIABLE FIELD
QUAL IS FOLLOWED BY BLANK
N
Y
LIBEND
BLPUNC2 UAAL
PUT LIB CARD
QUX
SAVE QUAL IN COMMUN. AREA
BDQUAL
MPUTD NONE
PUT DIAGNOSTIC NO. 2
SET IGNORE SWITCH
TO FINISH IN MACHINE OP ACBL
TO FINISH IN MACHINE OP ACBL
TO FINISH IN MACHINE OP ACBL

TABLE LOOKUP PSEUDO-OP ROUTINE

OP-CODE	GO-TO		OP-CODE	GO-TO	
WRONG MNEM.	OPERR	ABA5	END	END	AOA3
IGNORE,PASS1		ACB1	ENTRY	ENTRY	AJA2
EXTENDED BC		ABJ4	EQU,MAX,OR MIN	EQMANI	AKB1
EXTENDED BSR		ABJ3	EXTRN	EXTRN	ALA1
DCL	DCL		LIB	LIB	ADE1
CCW	CCW		LTORG	LTORG	AMA2
CNOP	CNOP	AFA1	ORG	ORG	AOA2
COM	COM	AHA4	QUAL	QUAL	ADE3
CSECT	CSECT	AGA3	START	START	AEA1
DC OR DS	DCDS	AIA2	TEQU	TEQU	AKA1
DSECT	DSECT	AGA2			

CHART AD. PSEUDO ROUTINE; LIB, QUAL SUBROUTINES

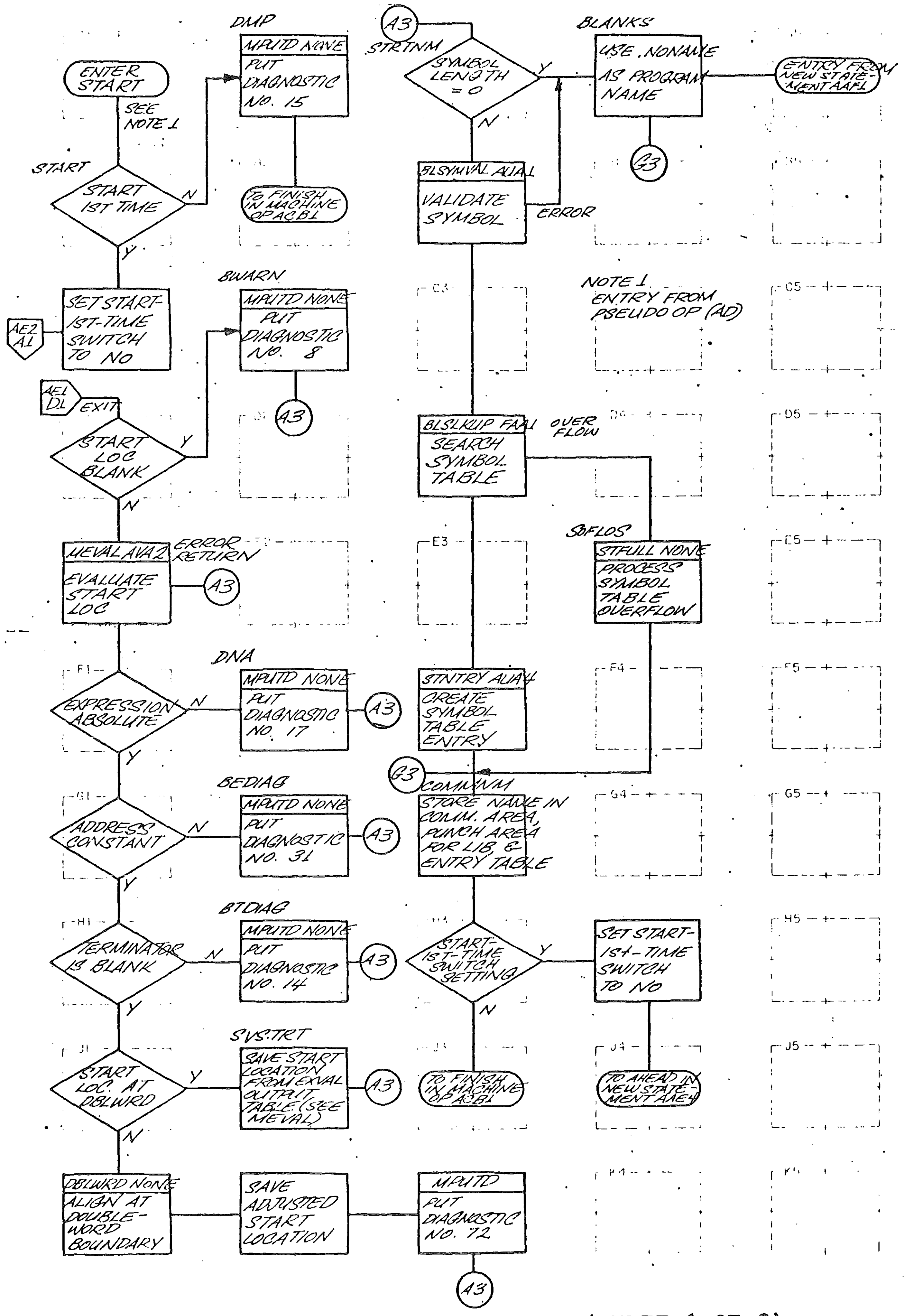


CHART AE1. START SUBROUTINE (SHEET 1 OF 2)

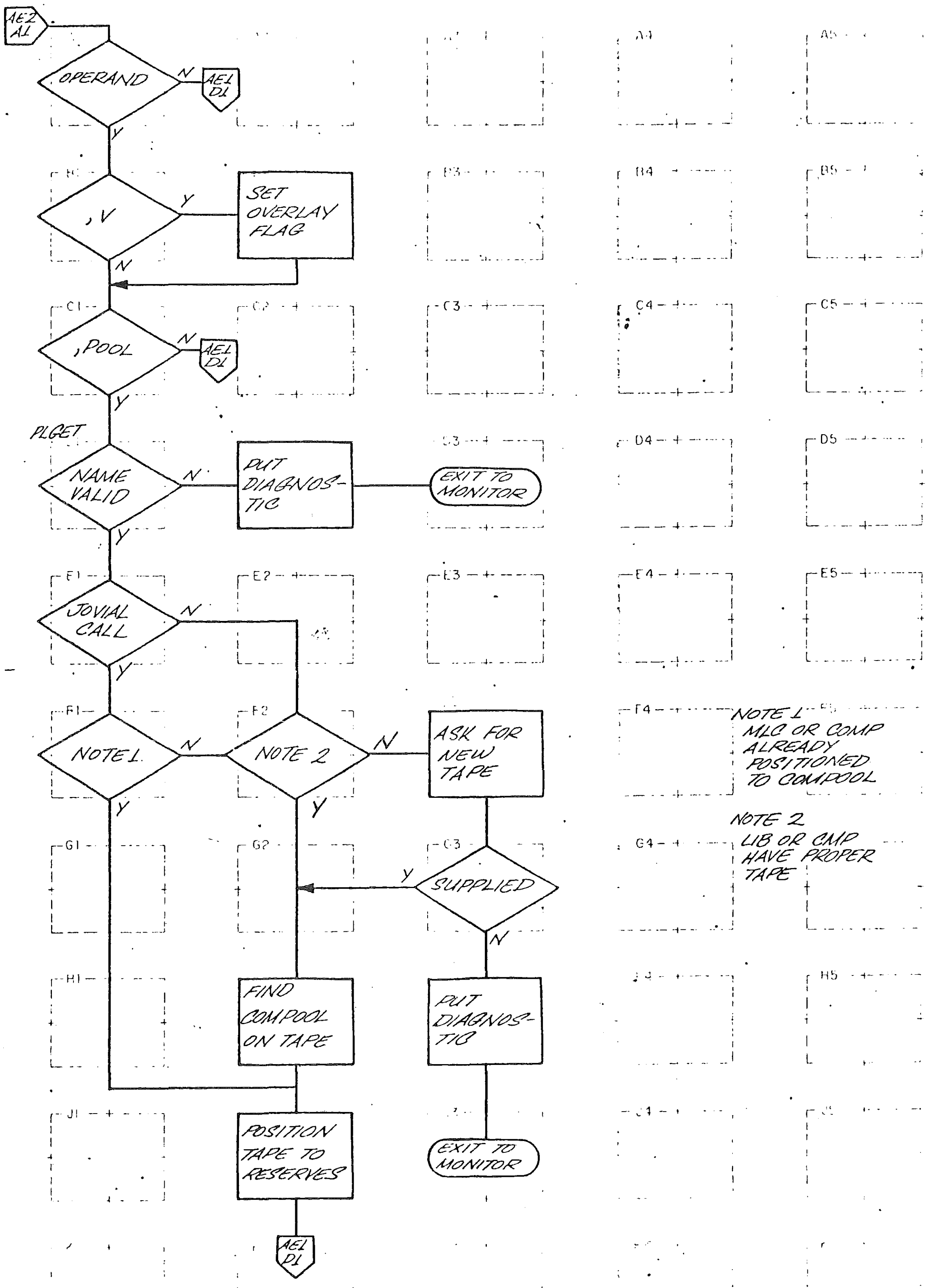


CHART AE2. START SUBROUTINE (SHEET 2 OF 2)

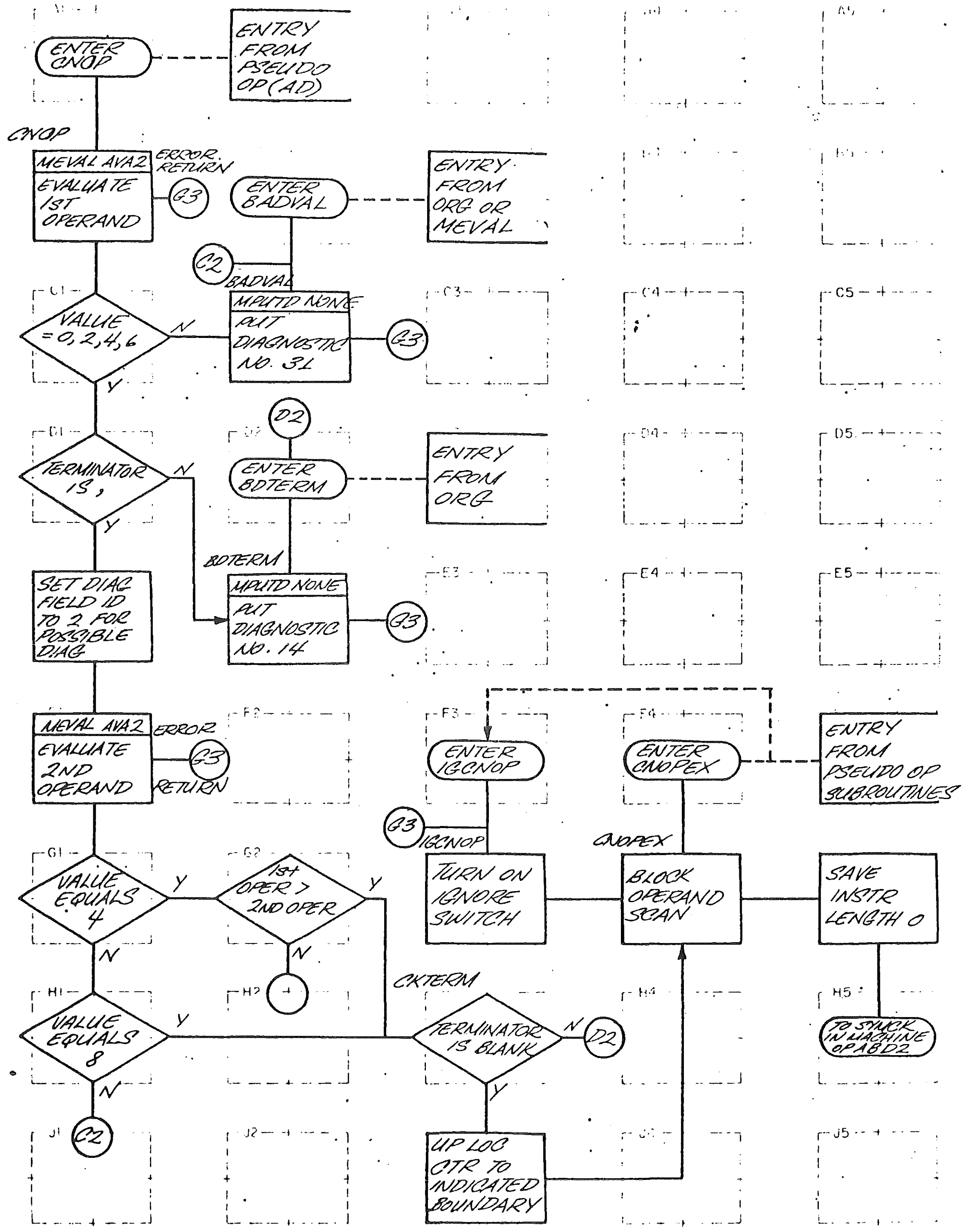


CHART AF. CNOP SUBROUTINE

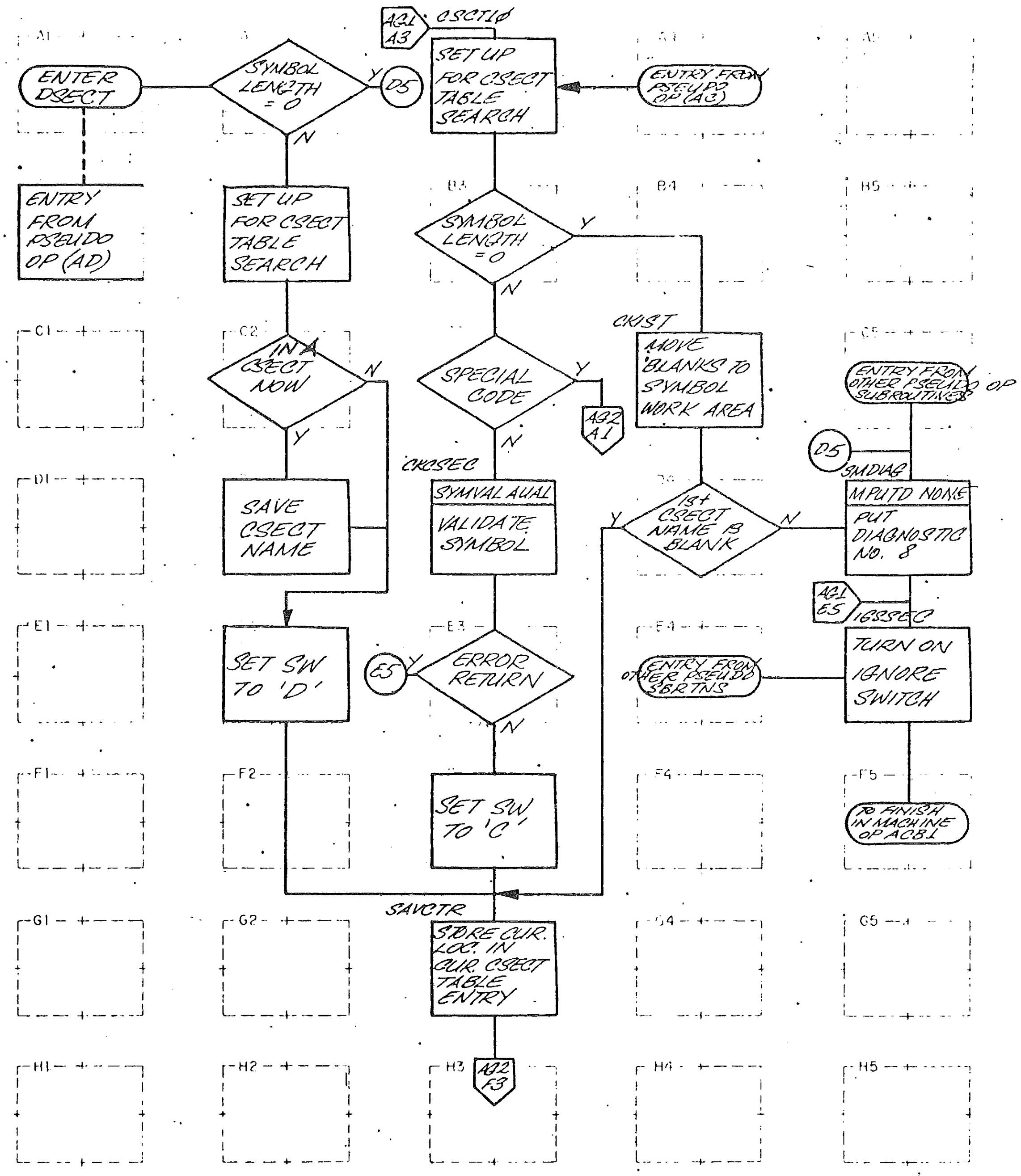


CHART AG1. CSECT, DSECT SUBROUTINES (SHEET 1 OF 3)

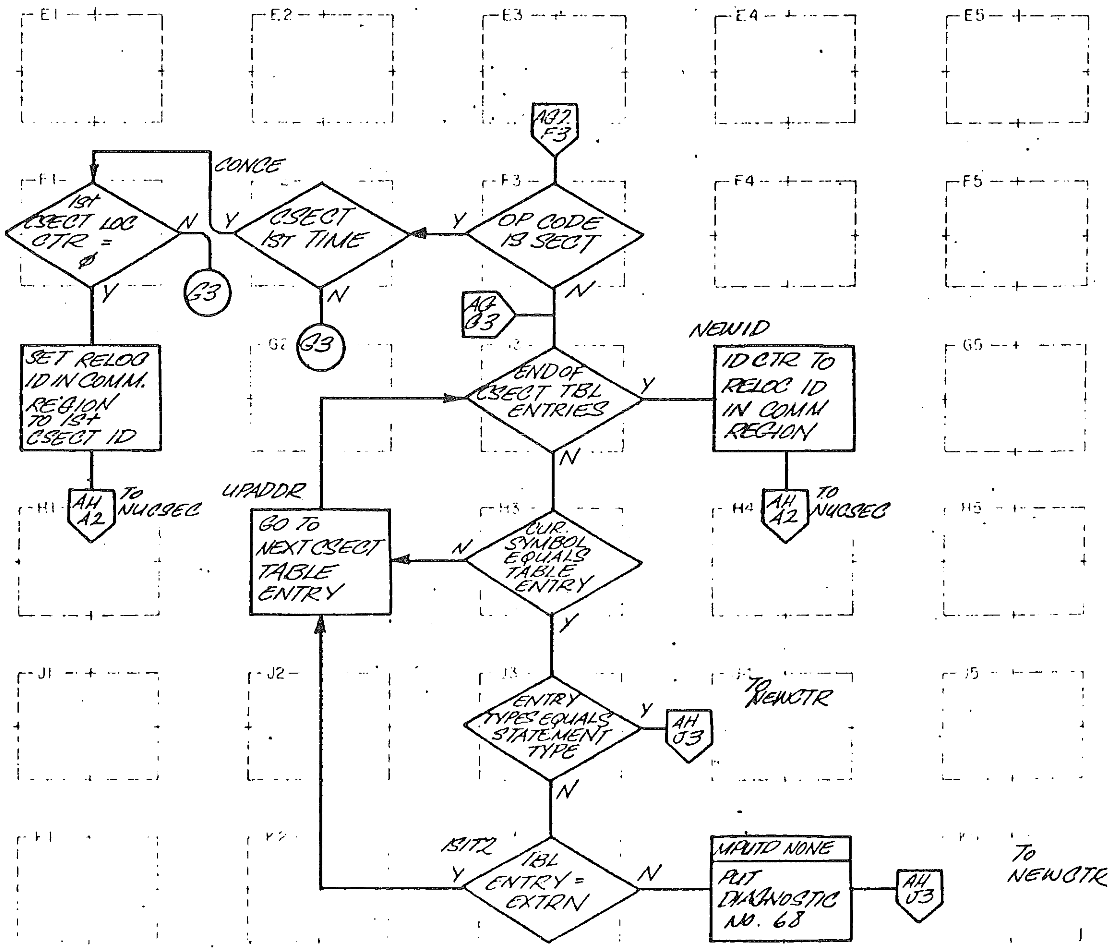


CHART AG2. CSECT, DSECT SUBROUTINES (SHEET 2 OF 3)

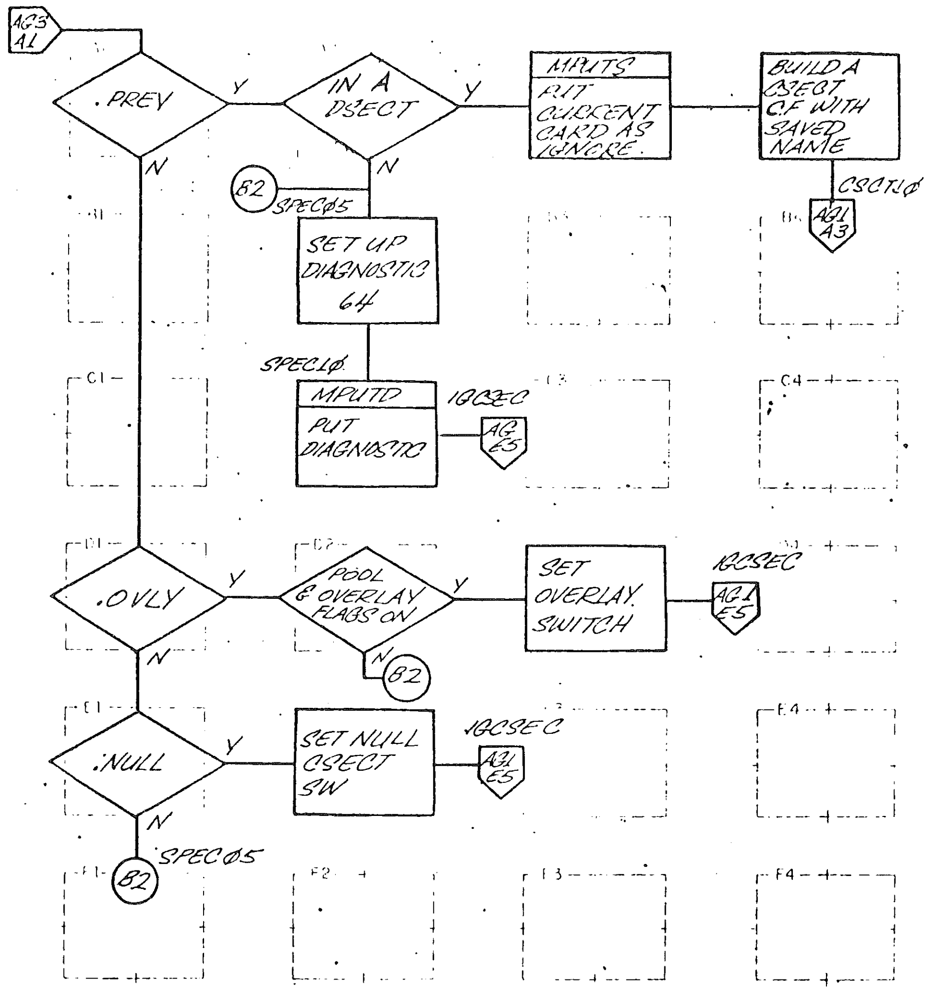


CHART AG3. CSECT, DSECT SUBROUTINES (SHEET 3 OF 3)

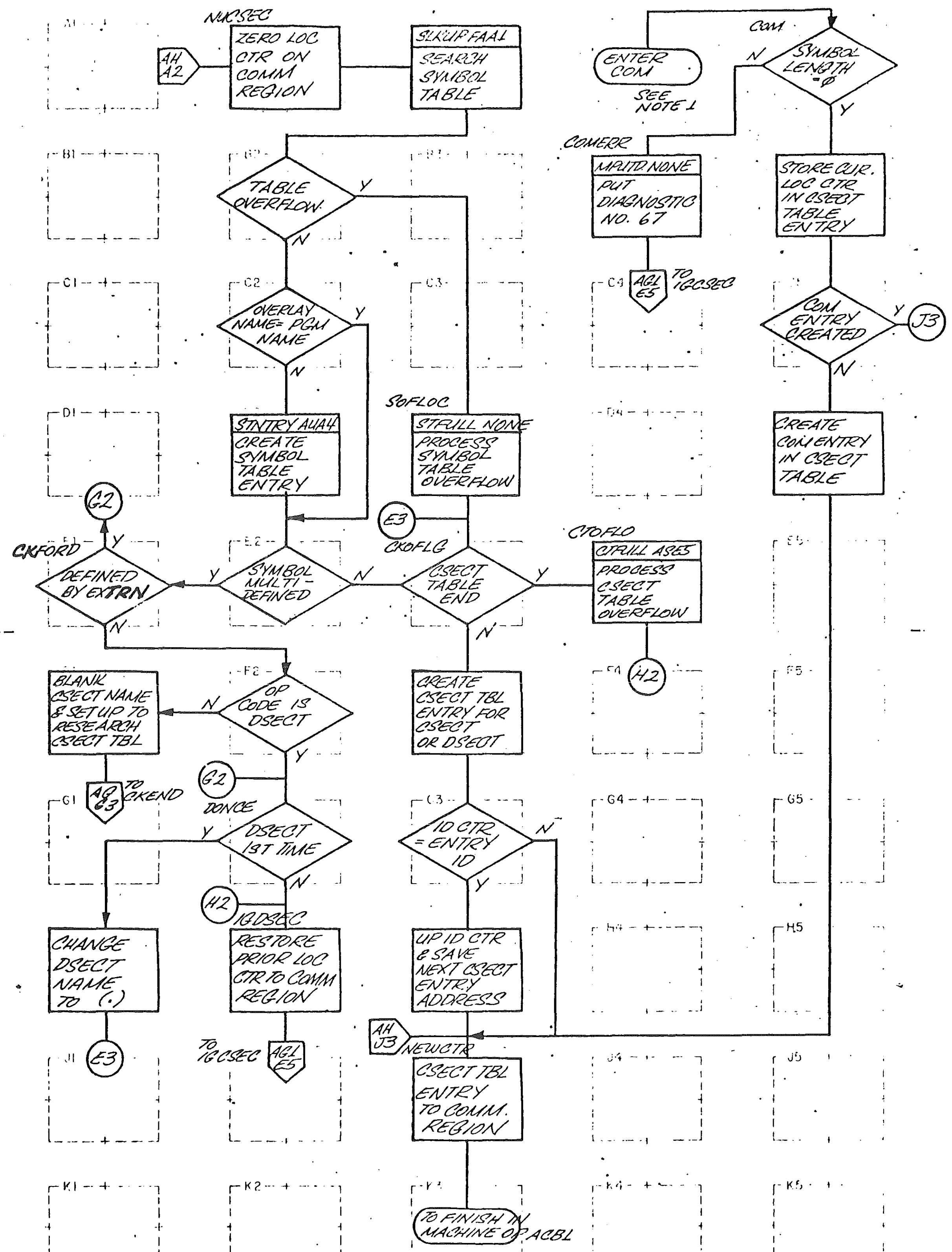


CHART AH. CSECT, DSECT, COM SUBROUTINES

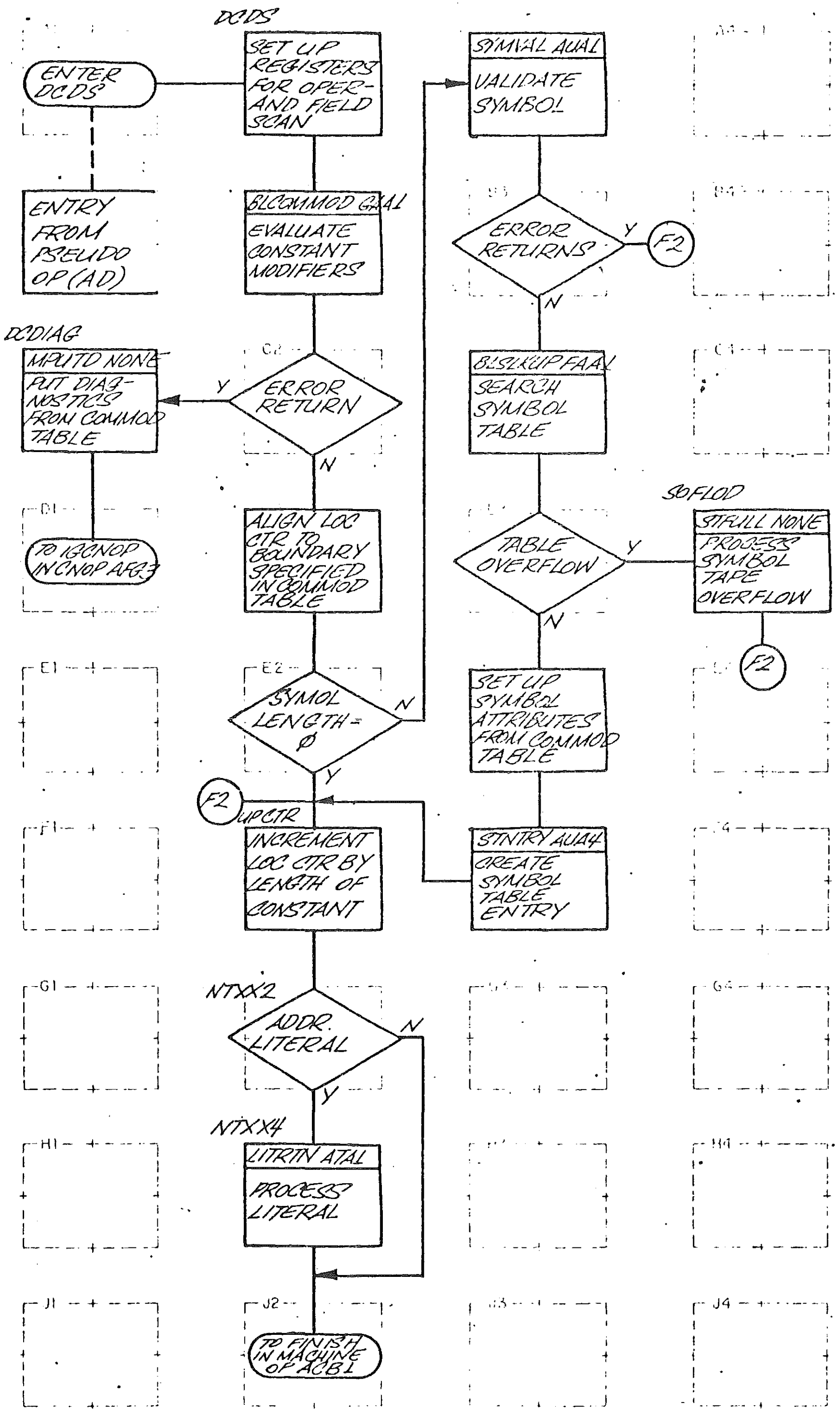

DCDS
ENTER DCDS
SET UP REGISTERS FOR OPER- AND FIELD SCAN
ENTRY FROM PSEUDO OP (AD)
BLCOMMOD GAAL
EVALUATE CONSTANT MODIFIERS
DCDIAG
MPUTD NONE
PUT DIAG-NOSTICS FROM COMMOD TABLE
Y
ERROR RETURN
N
TO IGCNOP IN CNOP AFG3
ALIGN LOC CTR TO BOUNDARY SPECIFIED IN COMMOD TABLE
SYMOL LENGTH = Ø
N
Y
F2
UPCTR
INCREMENT LOC CTR BY LENGTH OF CONSTANT
NTXX2
ADDR. LITERAL
N
Y
NTXX4
LITRTN ATAL
PROCESS LITERAL
TO FINISH IN MACHINE OF ACB1
SYMVAL AUAL
VALIDATE SYMBOL
ERROR RETURNS
Y
F2
N
BSLKUP FAAL
SEARCH SYMBOL TABLE
SOFLOD
TABLE OVERFLOW
Y
STFULL NONE
PROCESS SYMBOL TAPE OVERFLOW
F2
N
SET UP SYMBOL ATTRIBUTES FROM COMMOD TABLE
STNTRY AUA4
CREATE SYMBOL TABLE ENTRY

CHART AI. DCDS SUBROUTINE

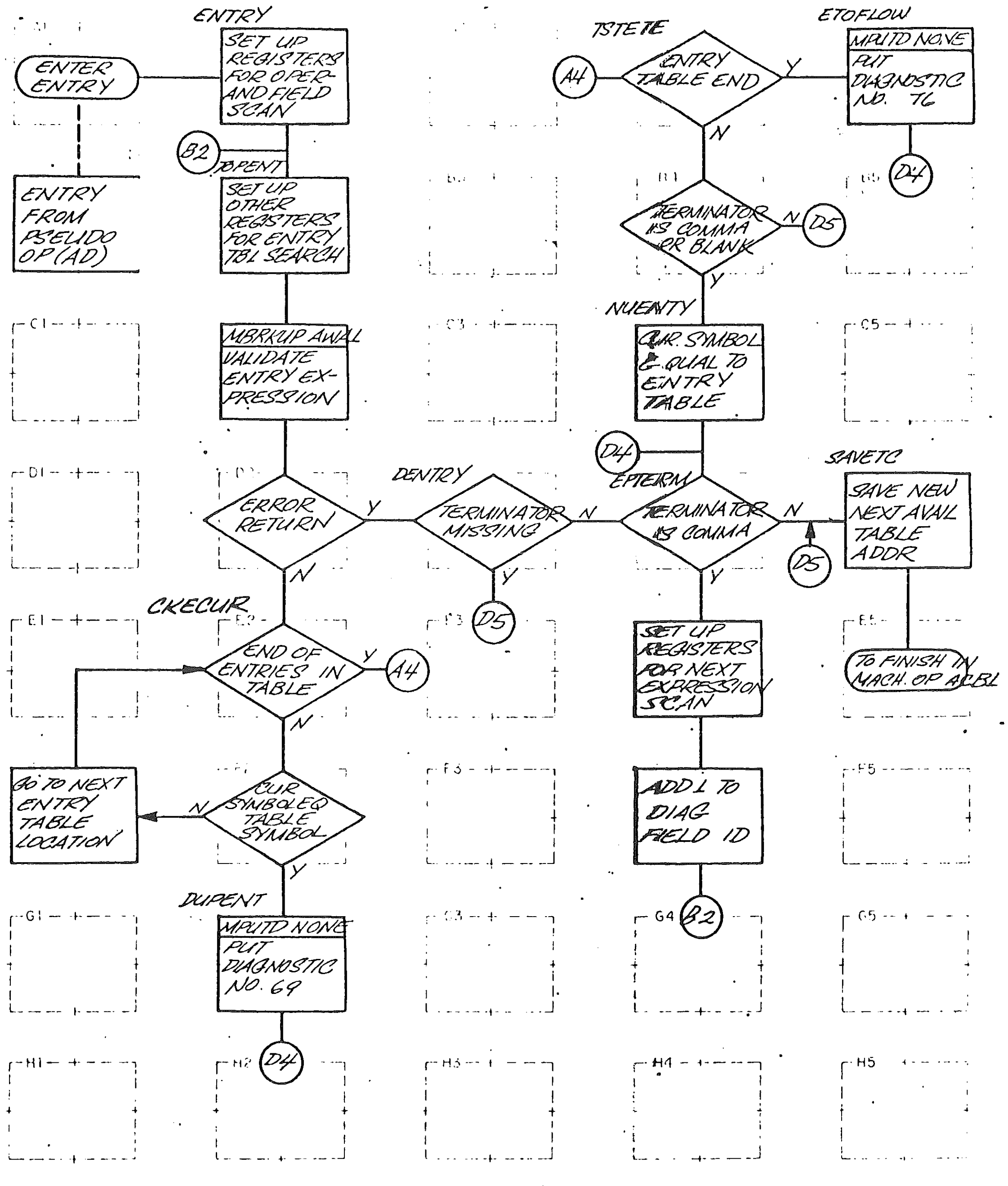

ENTER ENTRY
ENTRY FROM PSEUDO OP (AD)
ENTRY
SET UP REGISTERS FOR OPER- AND FIELD SCAN
B2
TOPENT
SET UP OTHER REGISTERS FOR ENTRY TBL SEARCH
MBRKUP AWALL
VALIDATE ENTRY EX-PRESSION
ERROR RETURN
Y
N
DENTRY
TERMINATOR MISSING
N
Y
D5
CKECUR
END OF ENTRIES IN TABLE
Y
A4
N
CUR SYMBOL EQ TABLE SYMBOL
N
GO TO NEXT ENTRY TABLE LOCATION
Y
DUPENT
MPUTD NONE
PUT DIAGNOSTIC NO. 69
D4
TSTETE
A4
ENTRY TABLE END
Y
N
ETOFLOW
MPUTD NONE
PUT DIAGNOSTIC NO. 76
D4
TERMINATOR IS COMMA OR BLANK
N
D5
Y
NUENTY
CUR. SYMBOL & QUAL TO ENTRY TABLE
D4
EPTERM
TERMINATOR IS COMMA
N
Y
D5
SAVETC
SAVE NEW NEXT AVAIL TABLE ADDR
TO FINISH IN MACH. OP ACBL
SET UP REGISTERS FOR NEXT EXPRESSION SCAN
ADD 1 TO DIAG FIELD ID
B2

CHART AJ. ENTRY SUBROUTINE

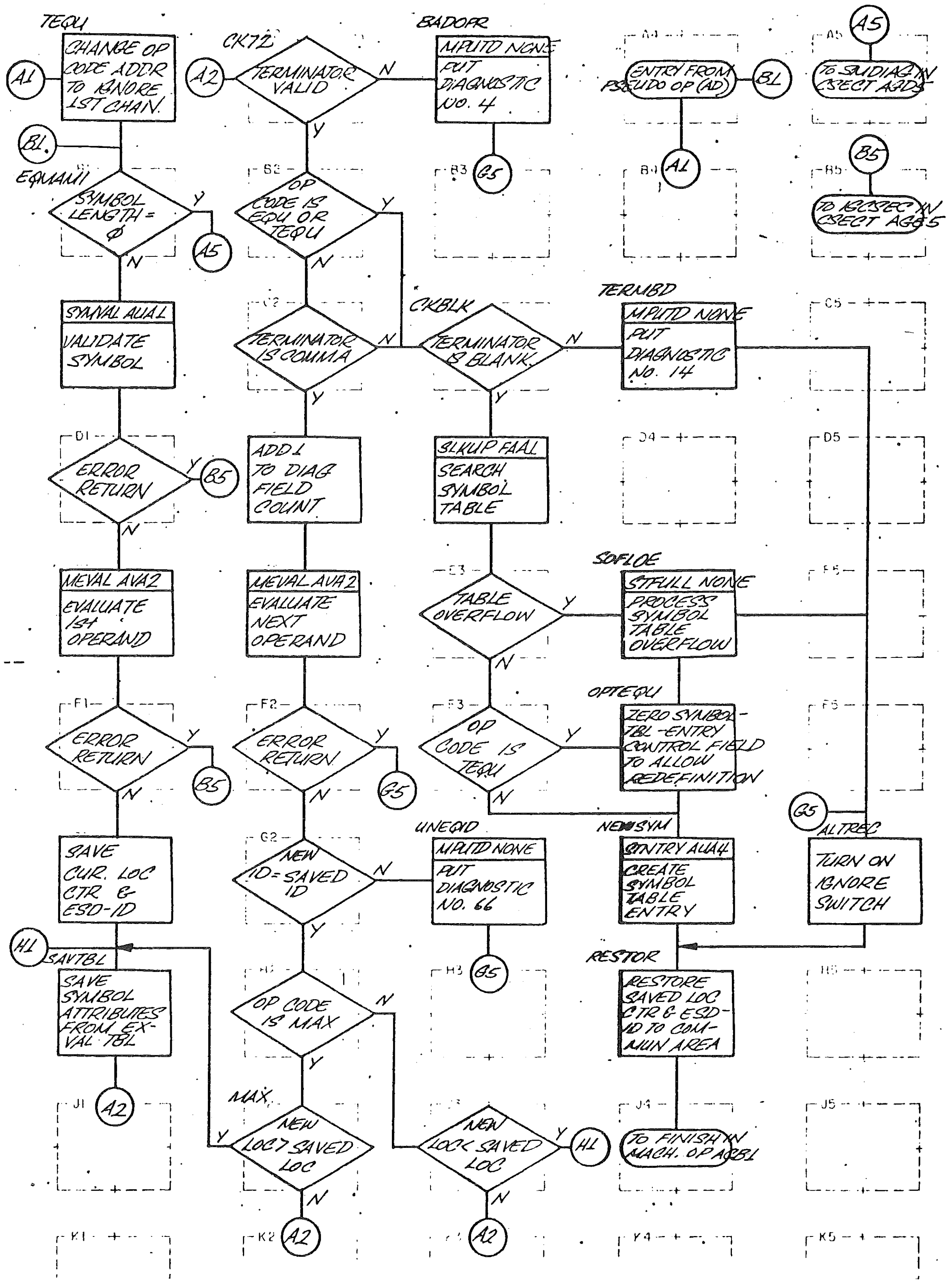


CHART AK. EQMAMI, TEQU SUBROUTINES

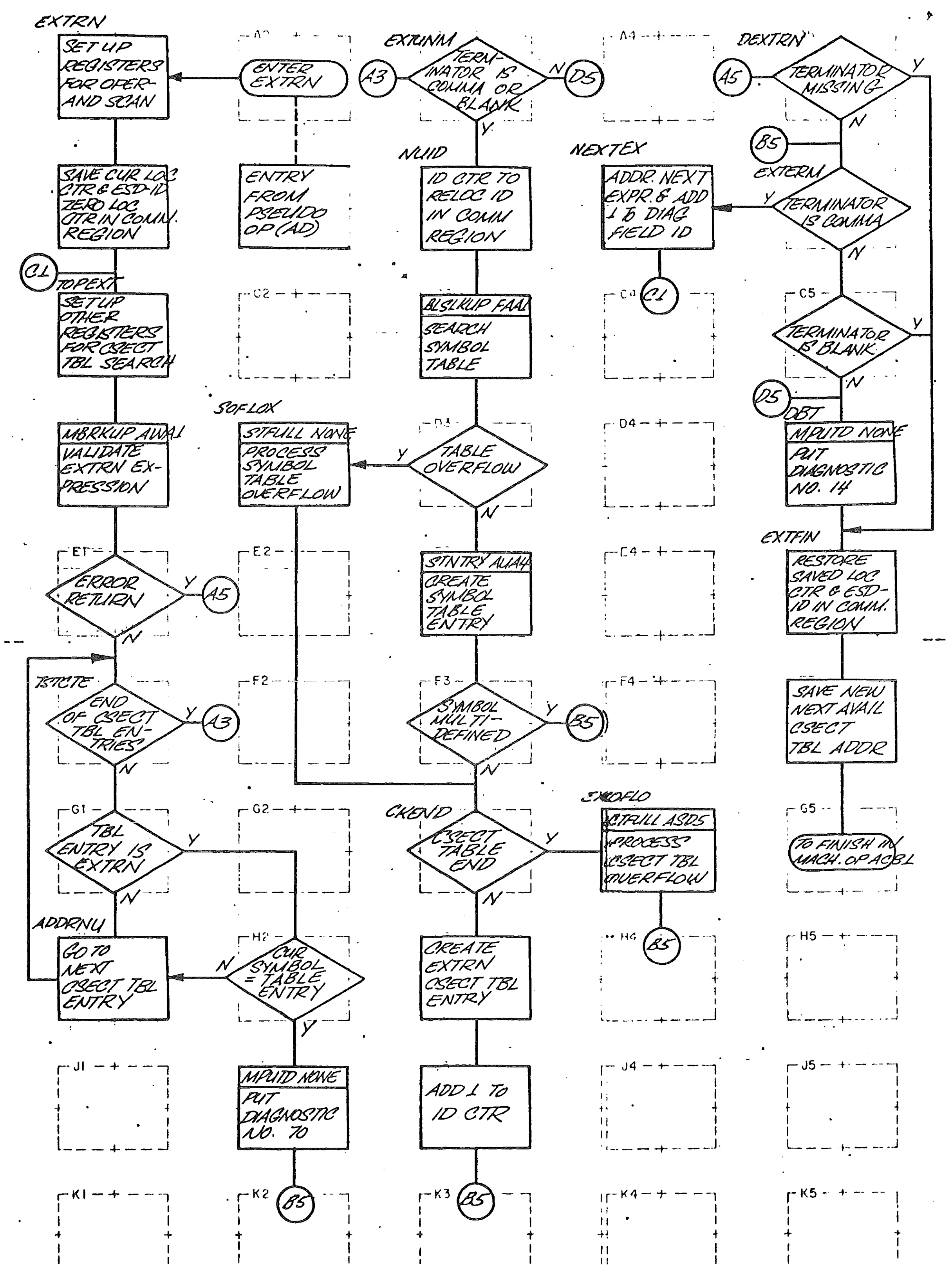
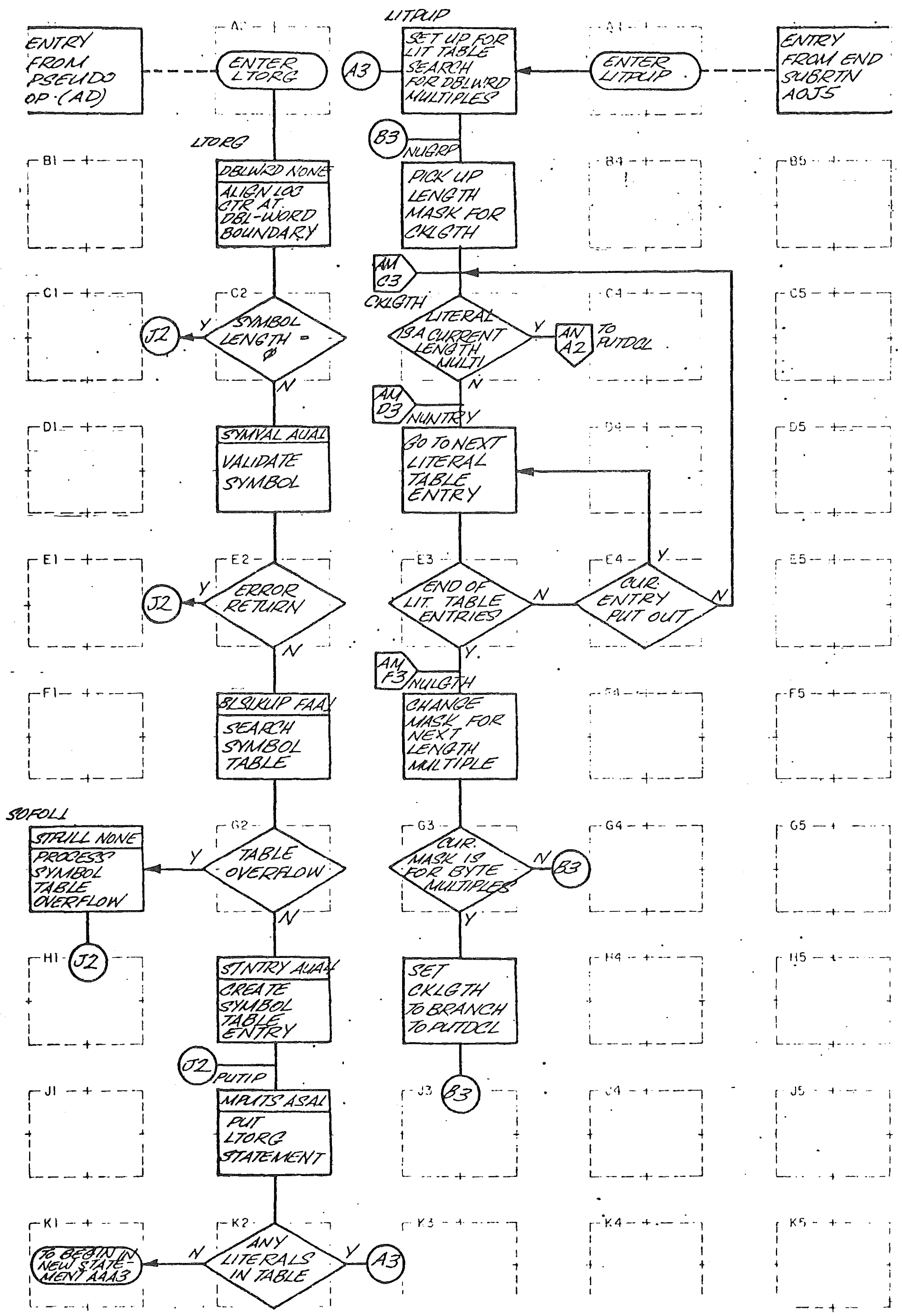

EXTRN
SET UP REGISTERS FOR OPERAND SCAN
ENTER EXTRN
ENTRY FROM PSEUDO OP (AD)
SAVE CUR LOC CTR & ESD-ID ZERO LOC CTR IN COMM. REGION
C1
TOPEXT
SET UP OTHER REGISTERS FOR CSECT TBL SEARCH
MBRKUP AWAL
VALIDATE EXTRN EXPRESSION
ERROR RETURN
A5
TSTCTE
END OF CSECT TBL ENTRIES
A3
TBL ENTRY IS EXTRN
ADDRNU
GO TO NEXT CSECT TBL ENTRY
CUR SYMBOL = TABLE ENTRY
MPUTD NONE
PUT DIAGNOSTIC NO. 70
B5
EXTRNM
TERMINATOR IS COMMA OR BLANK
D5
NUID
ID CTR TO RELOC ID IN COMM REGION
BLSLKUP FAAL
SEARCH SYMBOL TABLE
SOFLOX
STFULL NONE
PROCESS SYMBOL TABLE OVERFLOW
TABLE OVERFLOW
STNTRY AUA4
CREATE SYMBOL TABLE ENTRY
SYMBOL MULTI-DEFINED
CKEND
CSECT TABLE END
SYMOFLO
CTFULL ASDS
PROCESS CSECT TBL OVERFLOW
CREATE EXTRN CSECT TBL ENTRY
ADD 1 TO ID CTR
NEXTEX
ADDR NEXT EXPR & ADD 1 TO DIAG FIELD ID
DEXTRN
TERMINATOR MISSING
EXTERM
TERMINATOR IS COMMA
TERMINATOR IS BLANK
DBT
MPUTD NONE
PUT DIAGNOSTIC NO. 14
EXTFIN
RESTORE SAVED LOC CTR & ESD-ID IN COMM. REGION
SAVE NEW NEXT AVAIL CSECT TBL ADDR
TO FINISH IN MACH. OP ACBL
Y
N

CHART AL. EXTRN SUBROUTINE



ENTRY FROM PSEUDO OP (AD)
ENTER LTORG
LTORG
DBLWRD NONE
ALIGN LOC CTR AT DBL-WORD BOUNDARY
SYMBOL LENGTH = ∅
Y
N
J2
SYMVAL AUAL
VALIDATE SYMBOL
ERROR RETURN
BLSUKUP FAAY
SEARCH SYMBOL TABLE
TABLE OVERFLOW
SOFOLL
STFULL NONE
PROCESS SYMBOL TABLE OVERFLOW
STNTRY AUAY
CREATE SYMBOL TABLE ENTRY
PUTIP
MPUTS ASAL
PUT LTORG STATEMENT
ANY LITERALS IN TABLE
TO BEGIN IN NEW STATEMENT AAA3
A3
LITPUP
SET UP FOR LIT TABLE SEARCH FOR DBLWRD MULTIPLES
ENTER LITPUP
ENTRY FROM END SUBRTN AOJS
B3
NUGRP
PICK UP LENGTH MASK FOR CKLGTH
AM C3
CKLGTH
LITERAL IS A CURRENT LENGTH MULTI
AN A2
TO PUTDCL
AM D3
NUNTRY
GO TO NEXT LITERAL TABLE ENTRY
END OF LIT. TABLE ENTRIES
CUR. ENTRY PUT OUT
AM F3
NULGTH
CHANGE MASK FOR NEXT LENGTH MULTIPLE
CUR. MASK IS FOR BYTE MULTIPLES
SET CKLGTH TO BRANCH TO PUTDCL

CHART AM. LTORG SUBROUTINE (SHEET 1 OF 2)

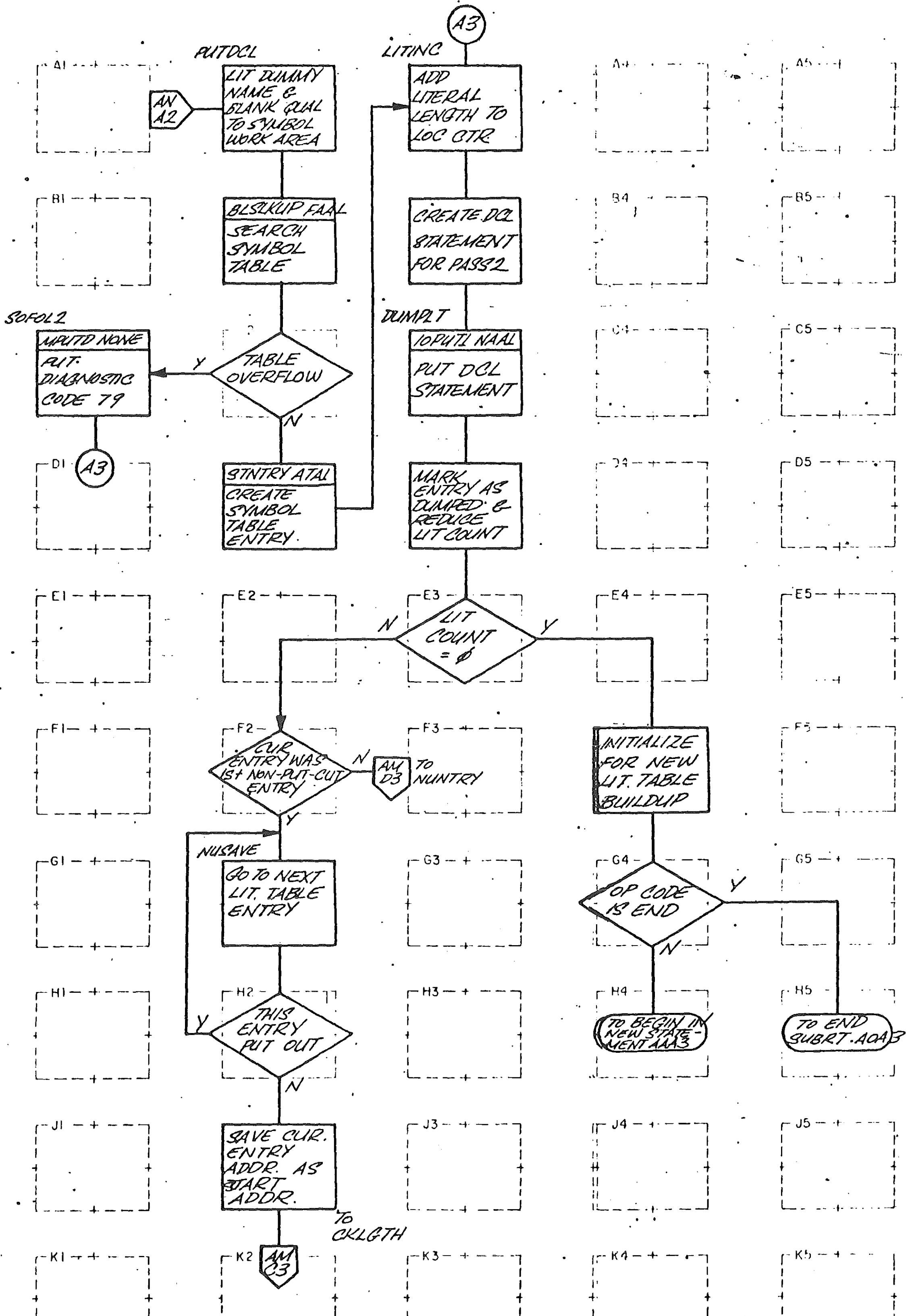

A3
PUTDCL
LIT DUMMY NAME & BLANK QUAL TO SYMBOL WORK AREA
AN A2
LITINC
ADD LITERAL LENGTH TO LOC CTR
BLSLKUP FAAL
SEARCH SYMBOL TABLE
CREATE DCL STATEMENT FOR PASS2
SOFOL2
MPUTD NONE
PUT DIAGNOSTIC CODE 79
Y
TABLE OVERFLOW
N
DUMPLT
IOPUTL NAAL
PUT DCL STATEMENT
A3
STNTRY ATAL
CREATE SYMBOL TABLE ENTRY
MARK ENTRY AS DUMPED & REDUCE LIT COUNT
N
LIT COUNT = ∅
Y
CUR. ENTRY WAS 1st NON-PUT-OUT ENTRY
N
AM D3
TO NUNTRY
INITIALIZE FOR NEW LIT. TABLE BUILDUP
Y
NUSAVE
GO TO NEXT LIT. TABLE ENTRY
OP CODE IS END
Y
N
Y
THIS ENTRY PUT OUT
TO BEGIN IN NEW STATEMENT AAA3
TO END SUBRT. AOA3
N
SAVE CUR. ENTRY ADDR. AS START ADDR.
TO CKLGTH
AM C3

CHART AN. LTORG SUBROUTINE (SHEET 2 OF 2)

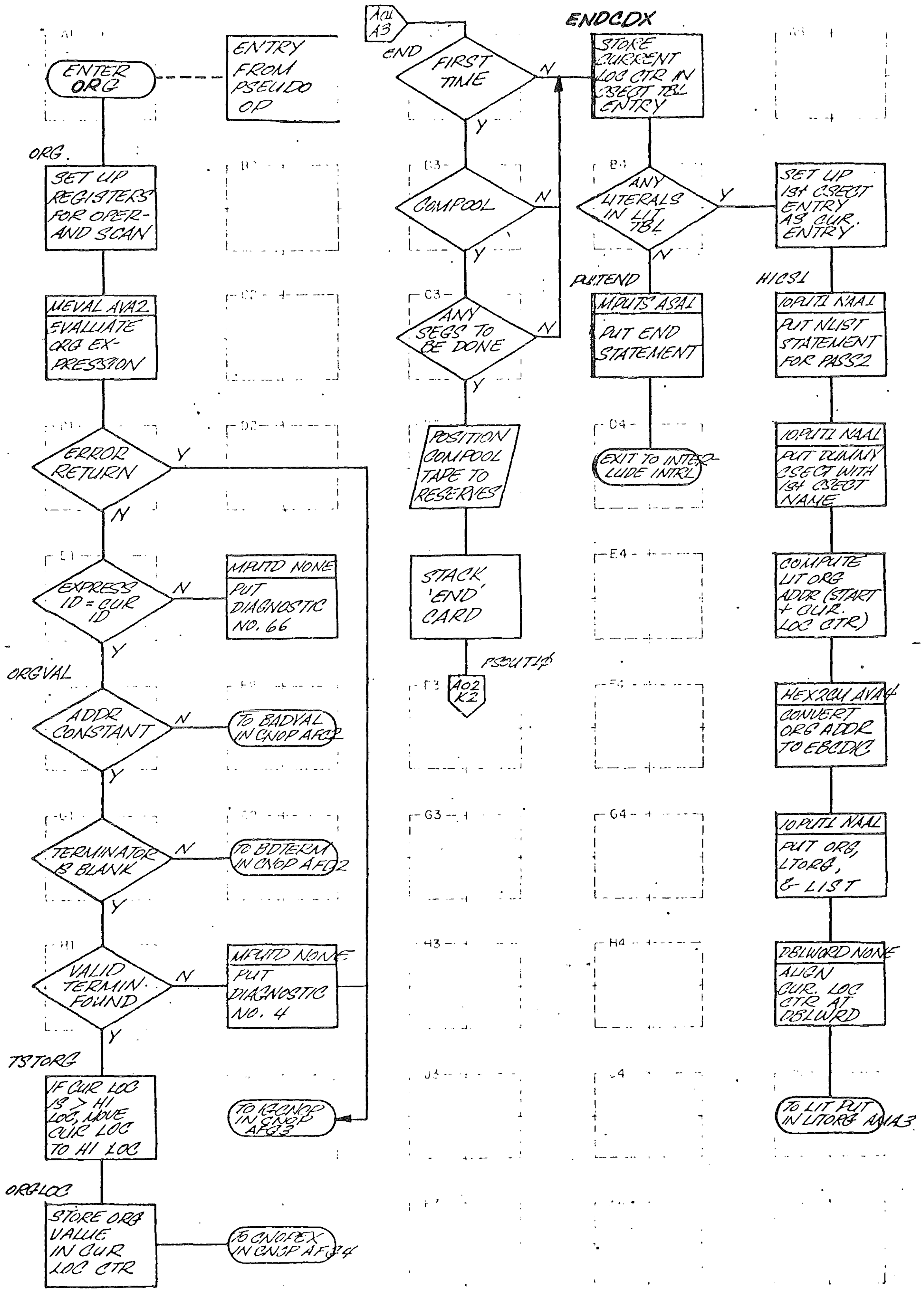


CHART A01. ORG, END SUBROUTINES (SHEET 1 OF 3)

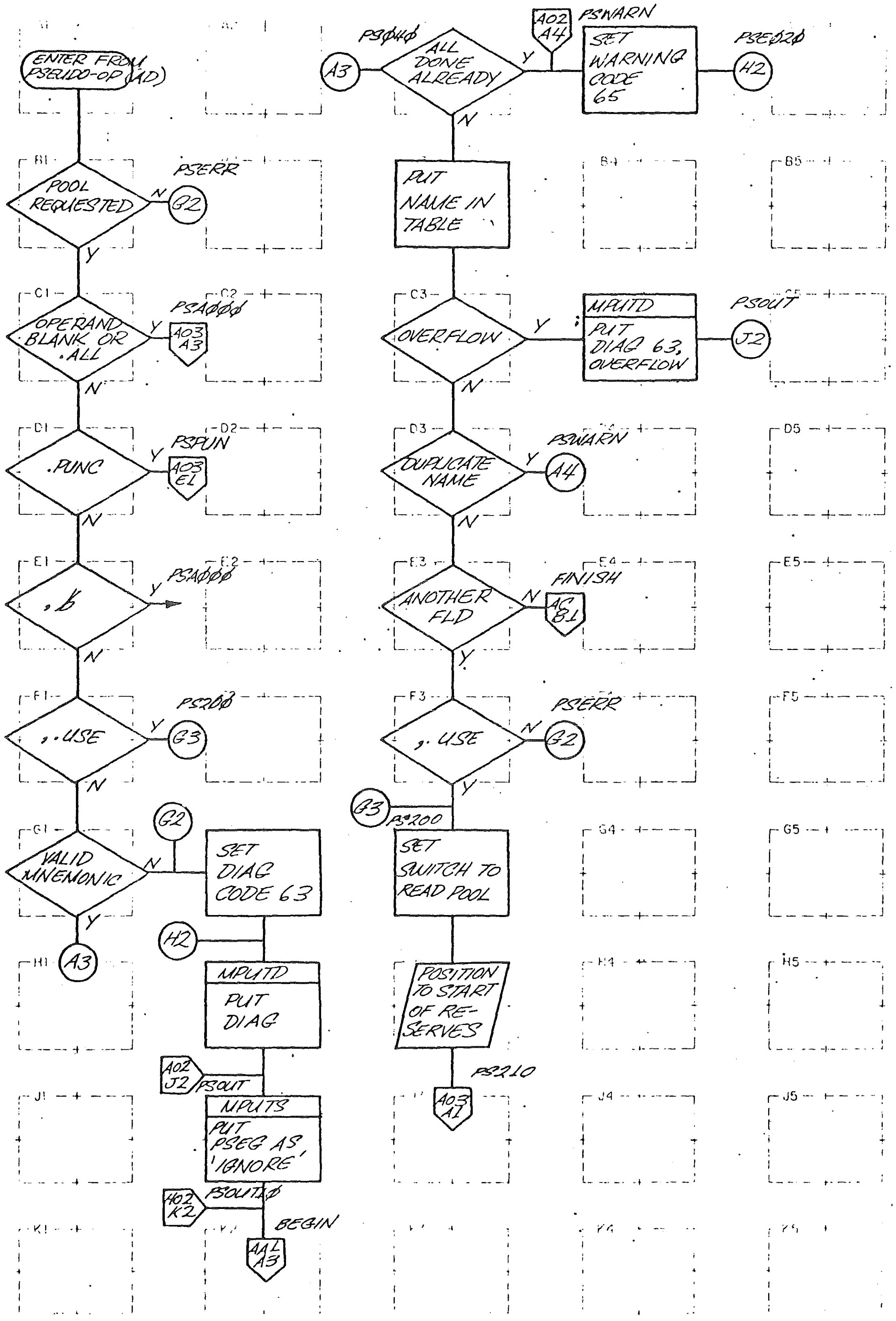


CHART AO2. ORG, END SUBROUTINES (SHEET 2 OF 3)

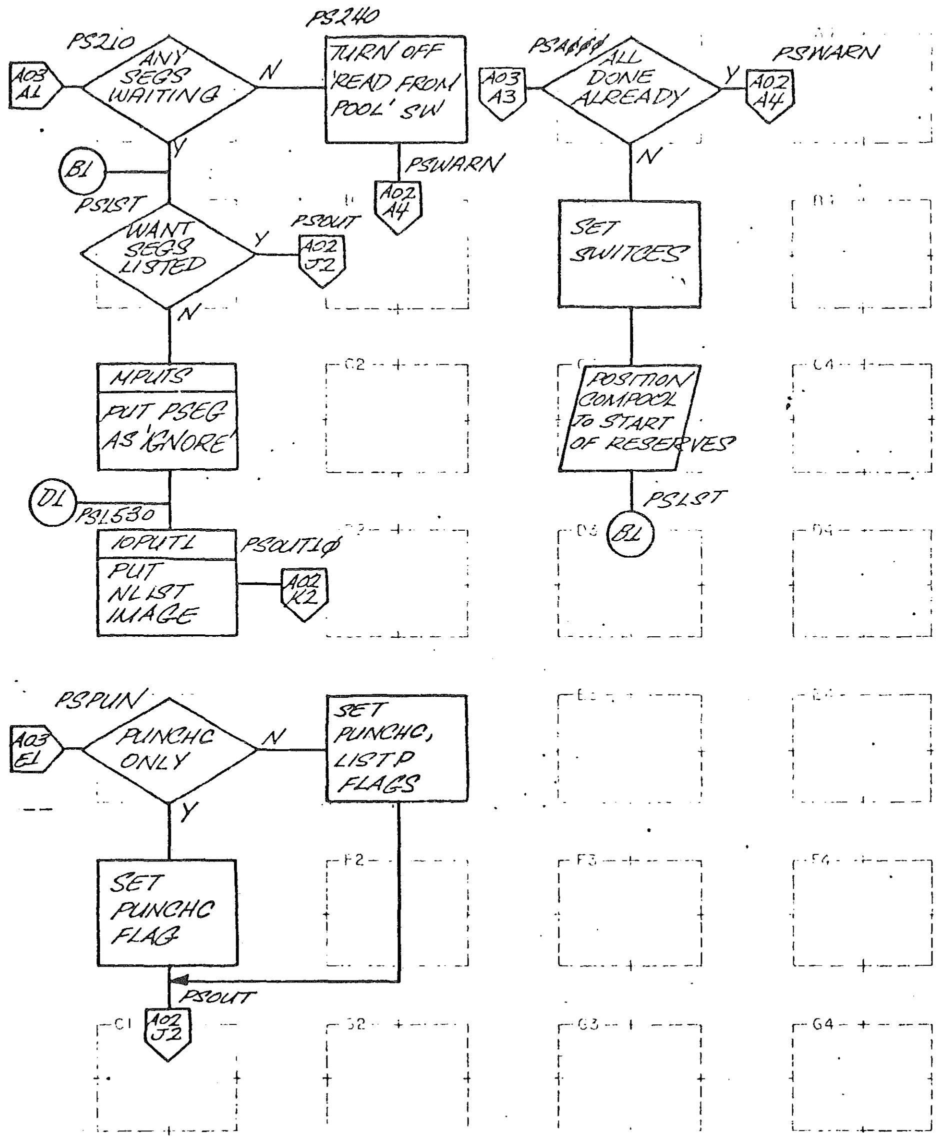
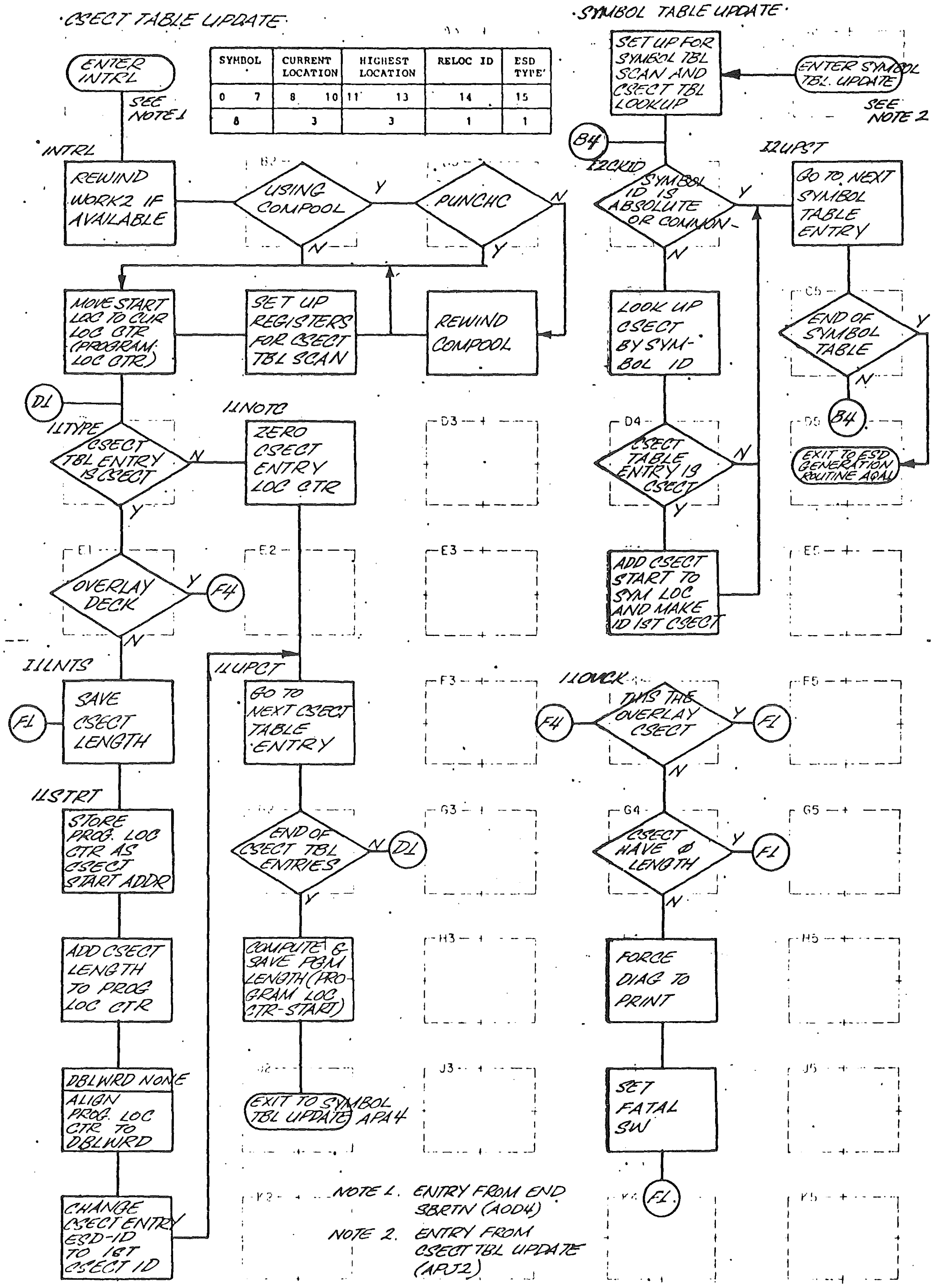


CHART AO3. ORG, END SUBROUTINES (SHEET 3 OF 3)


CSECT TABLE UPDATE
SYMBOL TABLE UPDATE
ENTER INTRL
SEE NOTE 1
INTRL
REWIND WORK2 IF AVAILABLE
USING COMPOOL
PUNCHC
Y
N
N
Y
MOVE START LOC TO CUR LOC CTR (PROGRAM LOC CTR)
SET UP REGISTERS FOR CSECT TBL SCAN
REWIND COMPOOL
D1
I1TYPE
CSECT TBL ENTRY IS CSECT
N
Y
I1NOTC
ZERO CSECT ENTRY LOC CTR
OVERLAY DECK
Y
F4
N
I1LNTS
SAVE CSECT LENGTH
F1
I1UPCT
GO TO NEXT CSECT TABLE ENTRY
I1STRT
STORE PROG. LOC CTR AS CSECT START ADDR
END OF CSECT TBL ENTRIES
N
D1
Y
ADD CSECT LENGTH TO PROG LOC CTR
COMPUTE & SAVE PGM LENGTH (PROGRAM LOC CTR-START)
DBLWRD NONE
ALIGN PROG. LOC CTR TO DBLWRD
EXIT TO SYMBOL TBL UPDATE APA4
CHANGE CSECT ENTRY ESD-ID TO 1ST CSECT ID
NOTE 1. ENTRY FROM END SBRTN (AOD4)
NOTE 2. ENTRY FROM CSECT TBL UPDATE (APJ2)
SET UP FOR SYMBOL TBL SCAN AND CSECT TBL LOOKUP
ENTER SYMBOL TBL. UPDATE
SEE NOTE 2
B4
I2CKID
SYMBOL ID IS ABSOLUTE OR COMMON
Y
N
I2UPST
GO TO NEXT SYMBOL TABLE ENTRY
LOOK UP CSECT BY SYMBOL ID
END OF SYMBOL TABLE
Y
N
B4
EXIT TO ESD GENERATION ROUTINE AQA1
CSECT TABLE ENTRY IS CSECT
N
Y
ADD CSECT START TO SYM LOC AND MAKE ID 1ST CSECT
I1OVCK
F4
THIS THE OVERLAY CSECT
Y
F1
N
CSECT HAVE Ø LENGTH
Y
F1
N
FORCE DIAG TO PRINT
SET FATAL SW
F1

SYMBOL	CURRENT LOCATION	HIGHEST LOCATION	RELOC ID	ESD TYPE
0 7	8 10	11 13	14	15
8	3	3	1	1

CHART AP. INTERLUDE-TABLES UPDATE

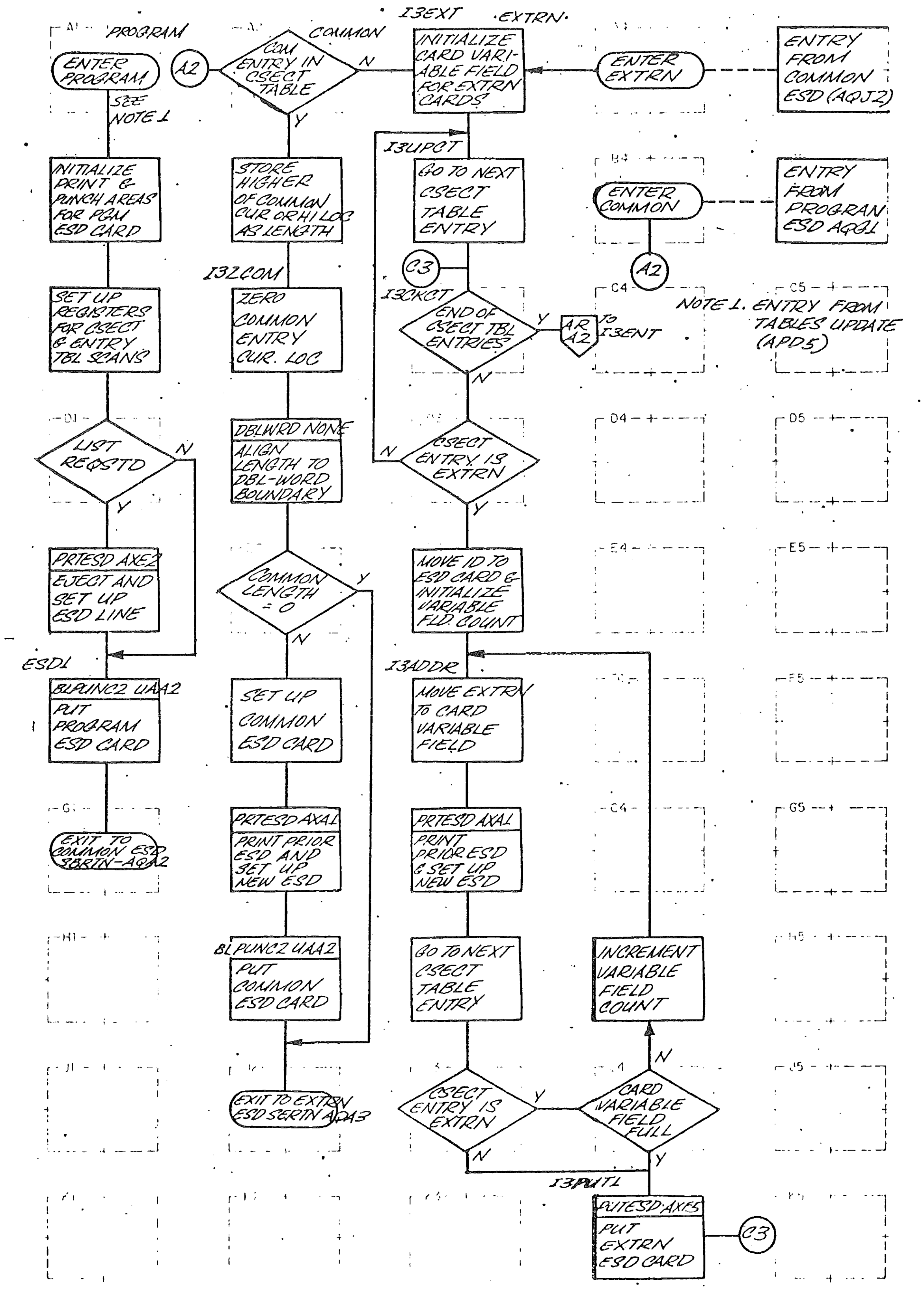

PROGRAM
ENTER PROGRAM
SEE NOTE 1
INITIALIZE PRINT & PUNCH AREAS FOR PGM ESD CARD
SET UP REGISTERS FOR CSECT & ENTRY TBL SCANS
LIST REQSTD
PRTESD AXE2
EJECT AND SET UP ESD LINE
ESD1
BLPUNC2 UAA2
PUT PROGRAM ESD CARD
EXIT TO COMMON ESD SERTN-AQA2
COMMON
COM ENTRY IN CSECT TABLE
STORE HIGHER OF COMMON CUR OR HI LOC AS LENGTH
I32COM
ZERO COMMON ENTRY CUR. LOC
DBLWRD NONE
ALIGN LENGTH TO DBL-WORD BOUNDARY
COMMON LENGTH = 0
SET UP COMMON ESD CARD
PRTESD AXA1
PRINT PRIOR ESD AND SET UP NEW ESD
BLPUNC2 UAA2
PUT COMMON ESD CARD
EXIT TO EXTRN ESD SERTN AQA3
I3EXT
.EXTRN.
INITIALIZE CARD VARIABLE FIELD FOR EXTRN CARDS
I3UPCT
GO TO NEXT CSECT TABLE ENTRY
I3CKCT
END OF CSECT TBL ENTRIES
AR A2
TO I3ENT
CSECT ENTRY IS EXTRN
MOVE ID TO ESD CARD & INITIALIZE VARIABLE FLD COUNT
I3ADDR
MOVE EXTRN TO CARD VARIABLE FIELD
PRTESD AXA1
PRINT PRIOR ESD & SET UP NEW ESD
GO TO NEXT CSECT TABLE ENTRY
CSECT ENTRY IS EXTRN
CARD VARIABLE FIELD FULL
INCREMENT VARIABLE FIELD COUNT
I3PUTL
PUTESD AXE5
PUT EXTRN ESD CARD
ENTER EXTRN
ENTRY FROM COMMON ESD (AQJ2)
ENTER COMMON
ENTRY FROM PROGRAM ESD AQG1
NOTE 1. ENTRY FROM TABLES UPDATE (APD5)
Y
N
A2
C3

CHART AQ. INTERLUDE-ESD GENERATOR (SHEET 1 OF 2)

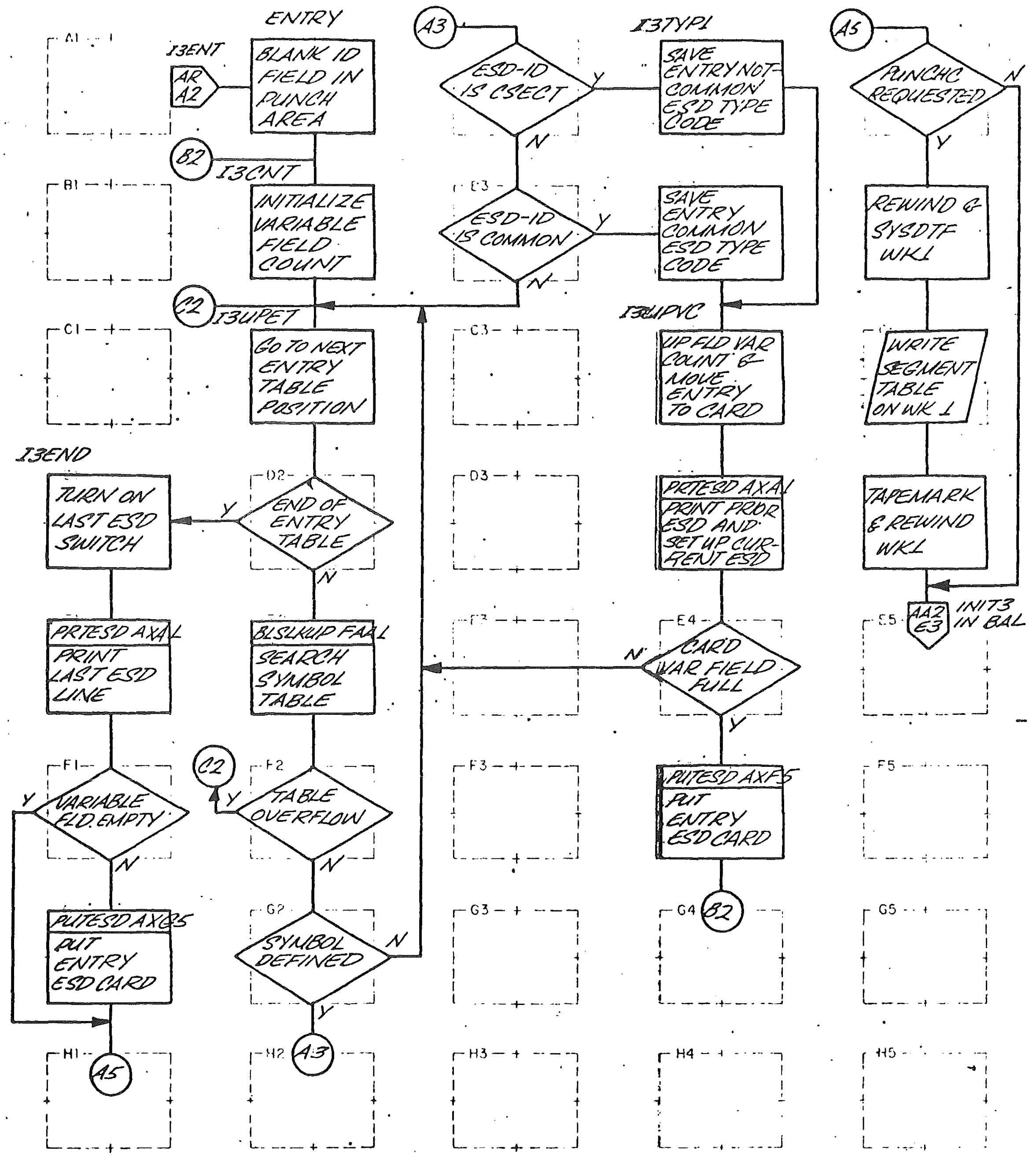


CHART AR. INTERLUDE-ESD GENERATOR (SHEET 2 OF 2)

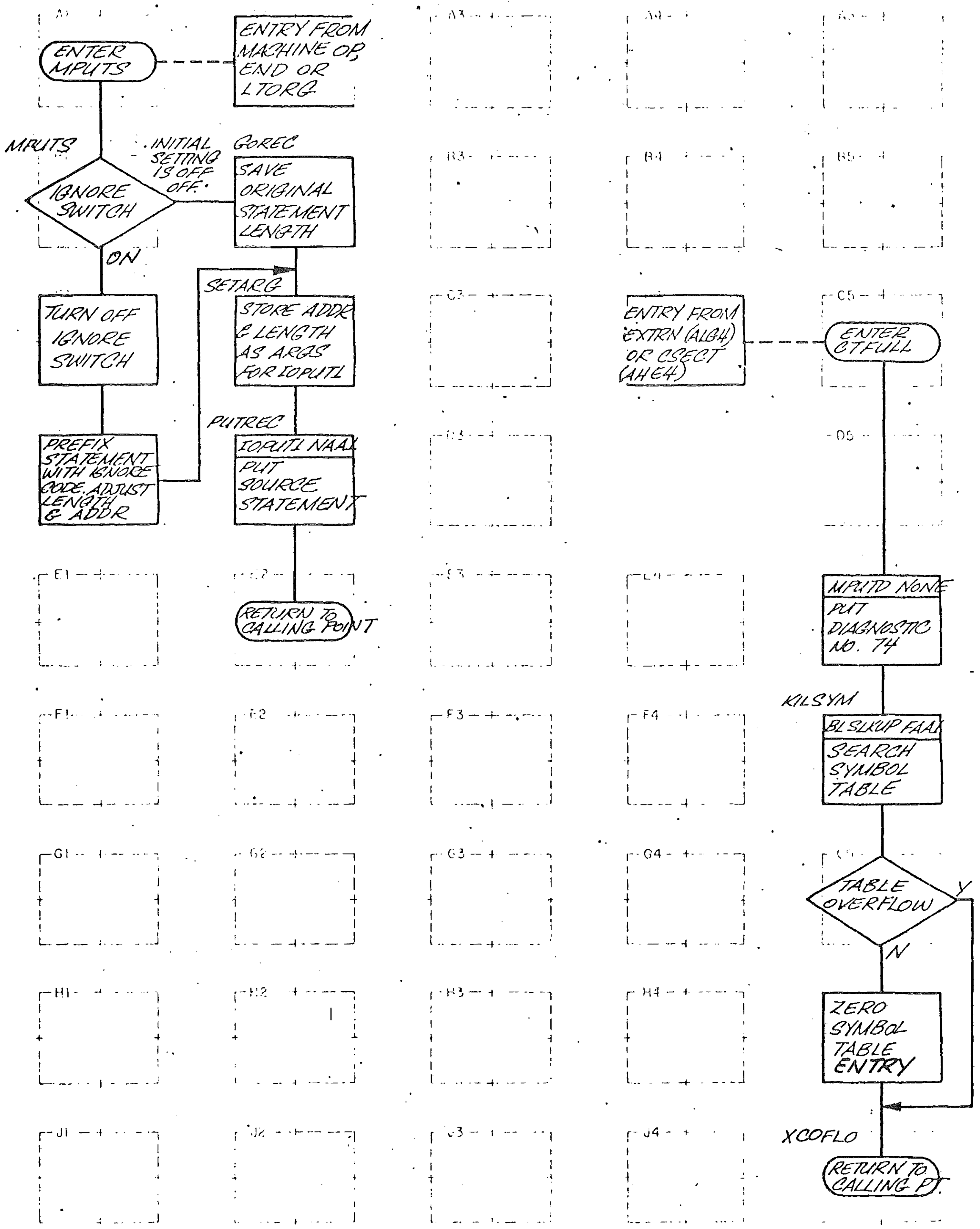


CHART AS. MPUTS, CTFULL SUBROUTINES

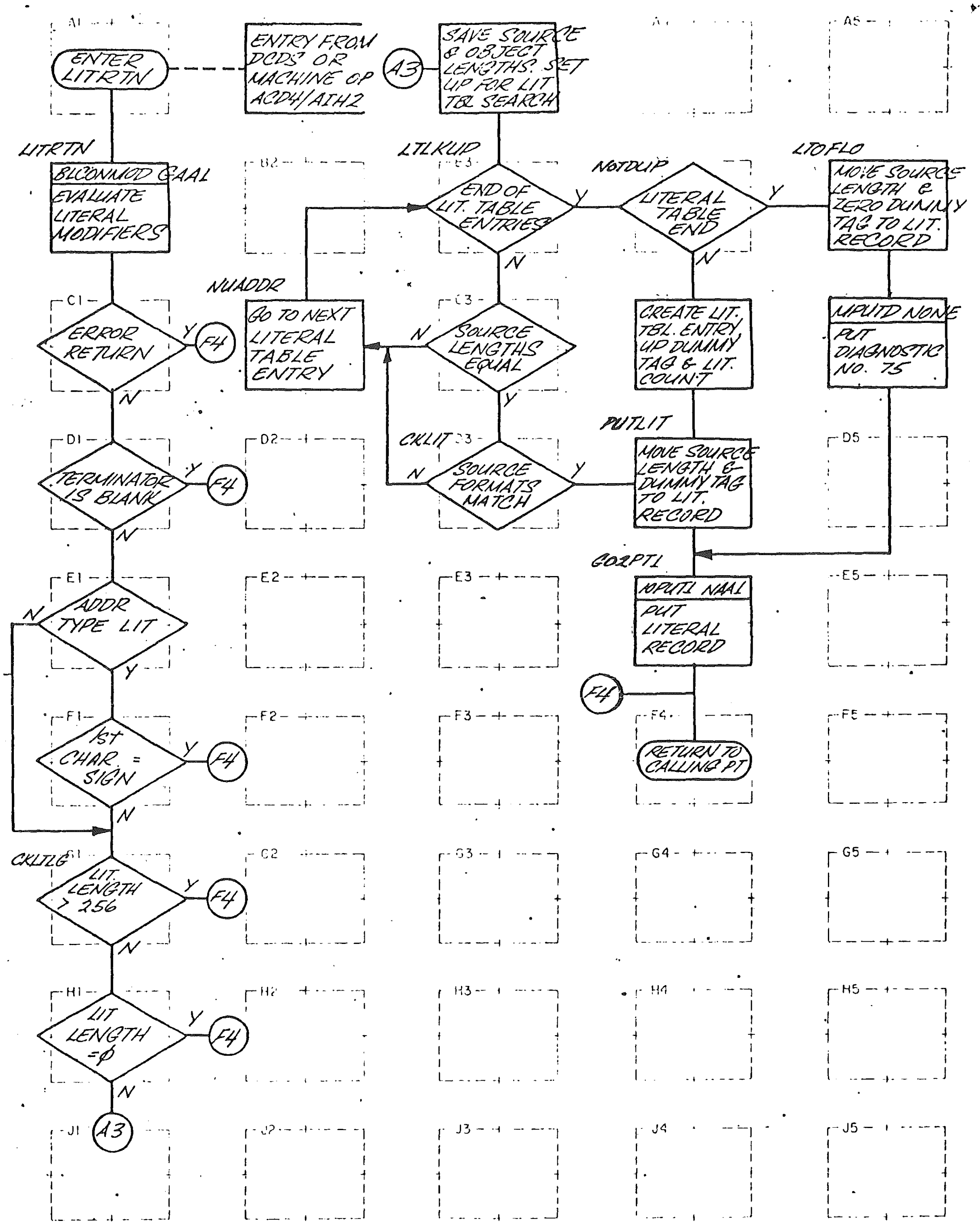

ENTER LITRTN
ENTRY FROM DCDS OR MACHINE OP ACD4/AIH2
A3
SAVE SOURCE & OBJECT LENGTHS. SET UP FOR LIT TBL SEARCH
LITRTN
BLCONMOD GAAL
EVALUATE LITERAL MODIFIERS
LTLKUP
END OF LIT. TABLE ENTRIES
NOTDUP
LITERAL TABLE END
LTOFLO
MOVE SOURCE LENGTH & ZERO DUMMY TAG TO LIT. RECORD
ERROR RETURN
F4
NUADDR
GO TO NEXT LITERAL TABLE ENTRY
SOURCE LENGTHS EQUAL
CREATE LIT. TBL. ENTRY, UP DUMMY TAG & LIT. COUNT
MPUTD NONE
PUT DIAGNOSTIC NO. 75
TERMINATOR IS BLANK
CKLIT
SOURCE FORMATS MATCH
PUTLIT
MOVE SOURCE LENGTH & DUMMY TAG TO LIT. RECORD
GO2PTL
ADDR TYPE LIT
MPUTL NAAL
PUT LITERAL RECORD
1st CHAR. = SIGN
RETURN TO CALLING PT
CKLTLG
LIT. LENGTH > 256
LIT LENGTH = ϕ
Y
N

CHART AT. LITRTN SUBROUTINE

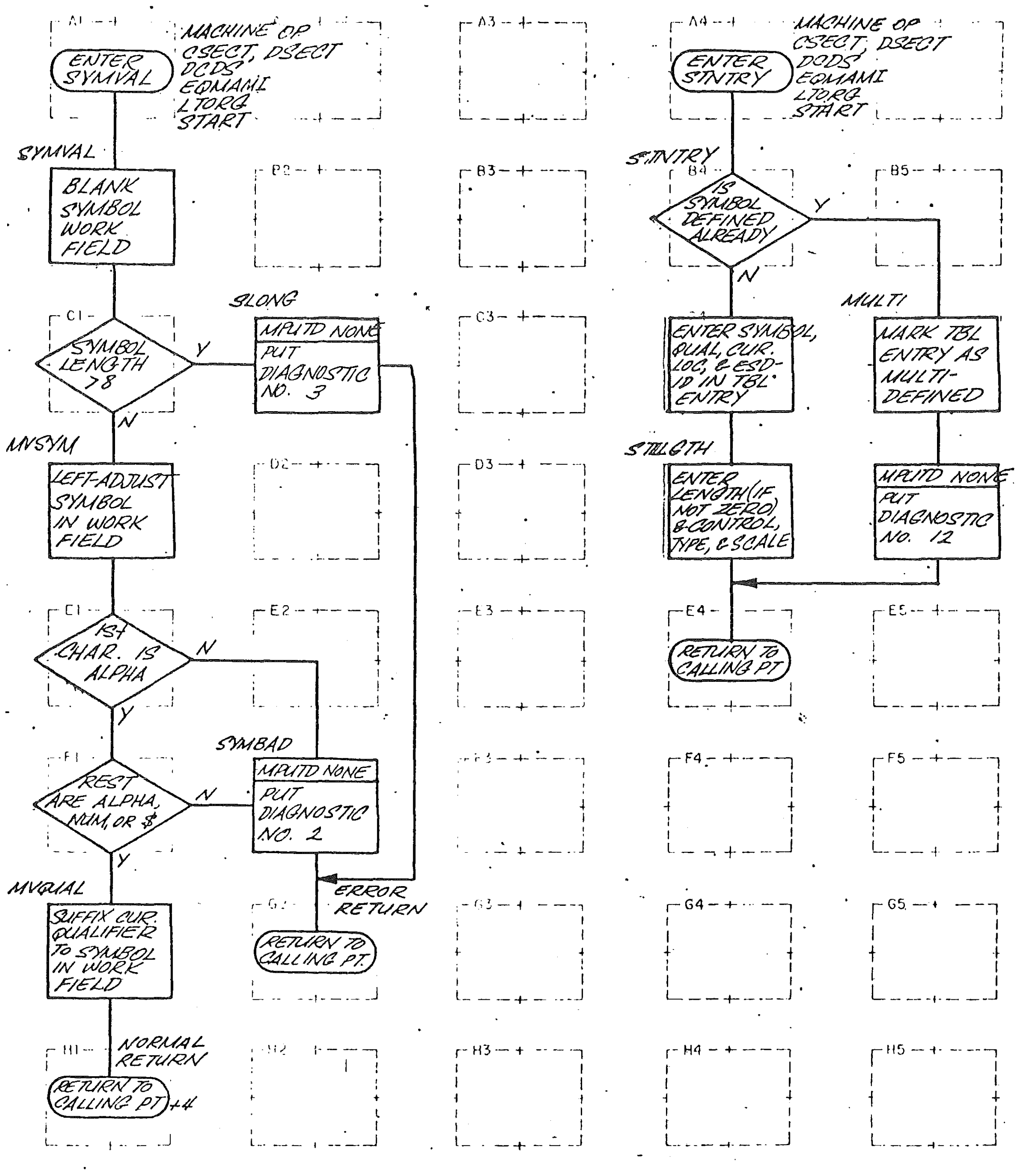

A1
ENTER SYMVAL
MACHINE OP CSECT, DSECT DC DS EQU AMI LTORG START
SYMVAL
BLANK SYMBOL WORK FIELD
C1
SYMBOL LENGTH >8
Y
N
SLONG
MPUTD NONE
PUT DIAGNOSTIC NO. 3
MVSYM
LEFT-ADJUST SYMBOL IN WORK FIELD
E1
1st CHAR. IS ALPHA
N
Y
SYMBAD
MPUTD NONE
PUT DIAGNOSTIC NO. 2
F1
REST ARE ALPHA, NUM, OR $
N
Y
MVQUAL
SUFFIX CUR. QUALIFIER TO SYMBOL IN WORK FIELD
ERROR RETURN
RETURN TO CALLING PT.
H1
NORMAL RETURN
RETURN TO CALLING PT +4
A4
ENTER STNTRY
MACHINE OP CSECT, DSECT DC DS EQU AMI LTORG START
STNTRY
B4
IS SYMBOL DEFINED ALREADY
Y
N
ENTER SYMBOL, QUAL, CUR. LOC, & ESD-ID IN TBL ENTRY
MULTI
MARK TBL ENTRY AS MULTI-DEFINED
STLGTH
ENTER LENGTH (IF NOT ZERO) & CONTROL, TYPE, & SCALE
MPUTD NONE
PUT DIAGNOSTIC NO. 12
E4
RETURN TO CALLING PT

CHART AU. SYMVAL, STNTRY SUBROUTINES

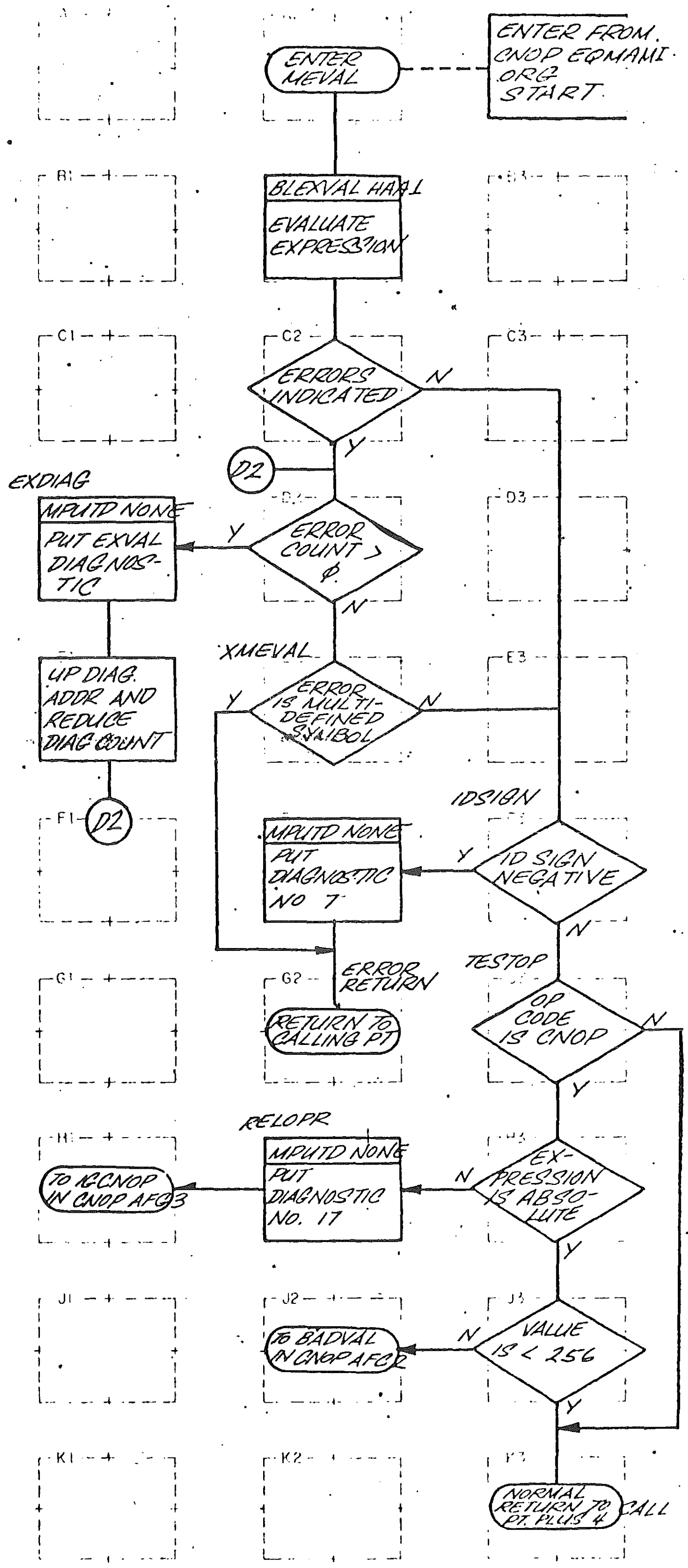


CHART AV. MEVAL SUBROUTINE

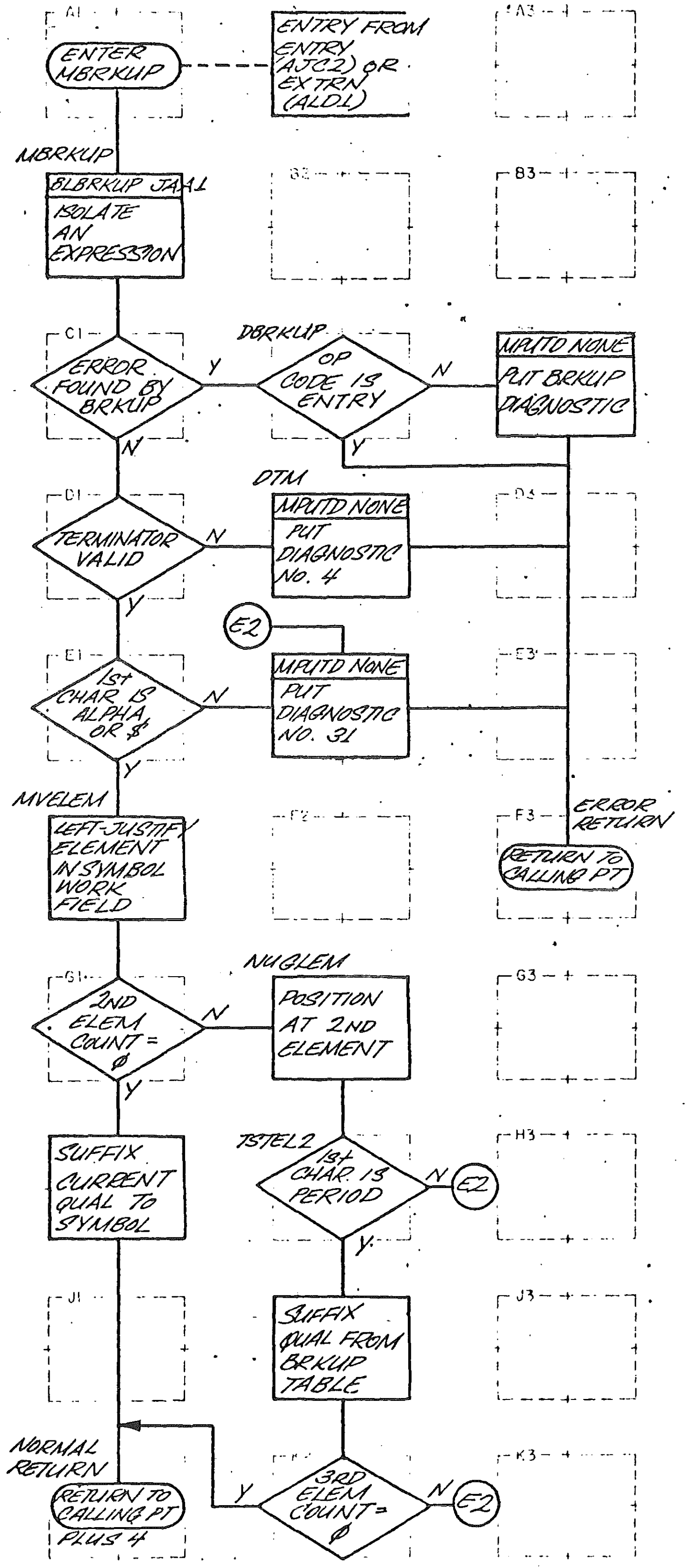


CHART AW. MBRKUP SUBROUTINE

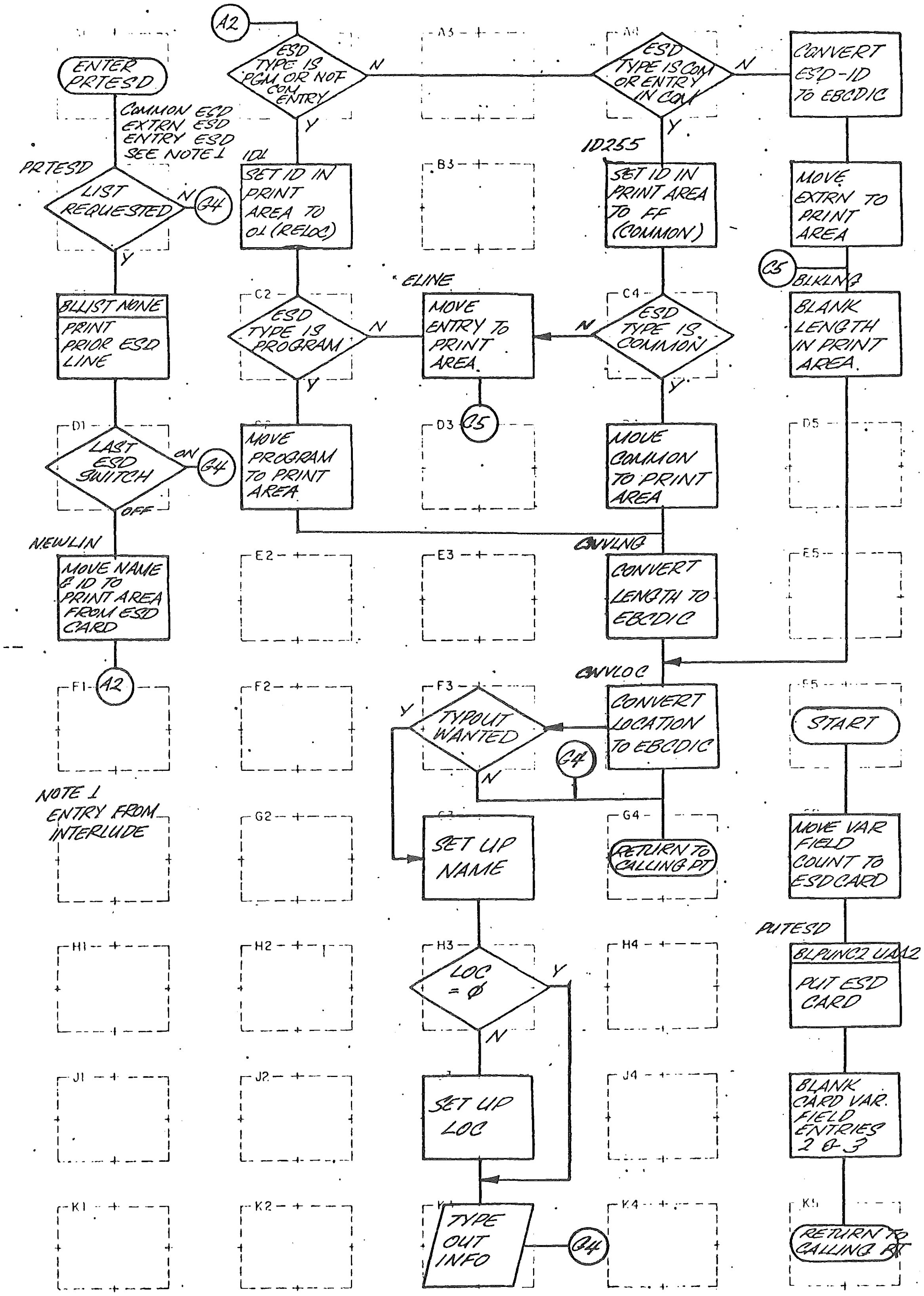

ENTER PRTESD
COMMON ESD EXTRN ESD ENTRY ESD SEE NOTE 1
PRTESD
LIST REQUESTED
N
G4
Y
BLLIST NONE
PRINT PRIOR ESD LINE
D1
LAST ESD SWITCH
ON
OFF
NEWLIN
MOVE NAME & ID TO PRINT AREA FROM ESD CARD
F1
A2
NOTE 1 ENTRY FROM INTERLUDE
ESD TYPE IS PGM OR NOT COM ENTRY
IDL
SET ID IN PRINT AREA TO 01 (RELOC)
C2
ESD TYPE IS PROGRAM
MOVE PROGRAM TO PRINT AREA
ELINE
MOVE ENTRY TO PRINT AREA
D3
C5
F3
TYPOUT WANTED
SET UP NAME
H3
LOC = Ø
SET UP LOC
TYPE OUT INFO
ESD TYPE IS COM OR ENTRY IN COM
ID255
SET ID IN PRINT AREA TO FF (COMMON)
C4
ESD TYPE IS COMMON
MOVE COMMON TO PRINT AREA
CNVLNG
CONVERT LENGTH TO EBCDIC
CNVLOC
CONVERT LOCATION TO EBCDIC
G4
RETURN TO CALLING PT
CONVERT ESD-ID TO EBCDIC
MOVE EXTRN TO PRINT AREA
BLKLNG
BLANK LENGTH IN PRINT AREA
START
MOVE VAR FIELD COUNT TO ESD CARD
PUTESD
BLPUNC2 UA12
PUT ESD CARD
BLANK CARD VAR. FIELD ENTRIES 2 & 3
RETURN TO CALLING PT

CHART AX. PUTESD, PRTESD SUBROUTINES

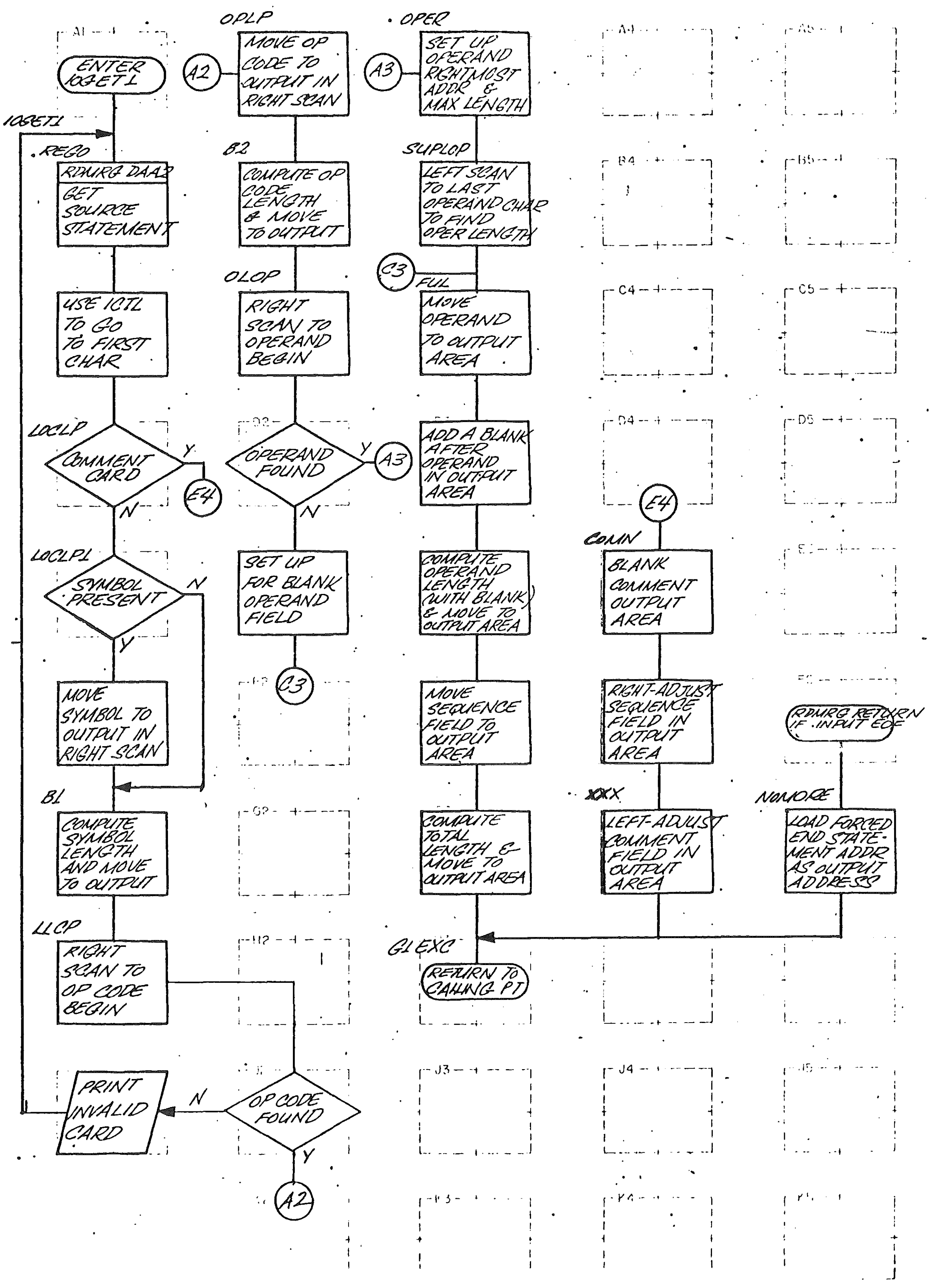

ENTER IOGET1
IOGET1
.REGO
RDURG DAA2
GET SOURCE STATEMENT
USE ICTL TO GO TO FIRST CHAR
LOCLP
COMMENT CARD
Y
E4
N
LOCLP1
SYMBOL PRESENT
N
Y
MOVE SYMBOL TO OUTPUT IN RIGHT SCAN
B1
COMPUTE SYMBOL LENGTH AND MOVE TO OUTPUT
LLCP
RIGHT SCAN TO OP CODE BEGIN
PRINT INVALID CARD
N
OP CODE FOUND
Y
A2
OPLP
A2
MOVE OP CODE TO OUTPUT IN RIGHT SCAN
B2
COMPUTE OP CODE LENGTH & MOVE TO OUTPUT
OLOP
RIGHT SCAN TO OPERAND BEGIN
OPERAND FOUND
Y
A3
N
SET UP FOR BLANK OPERAND FIELD
C3
.OPER
A3
SET UP OPERAND RIGHTMOST ADDR & MAX LENGTH
SUPLOP
LEFT SCAN TO LAST OPERAND CHAR TO FIND OPER LENGTH
C3
FUL
MOVE OPERAND TO OUTPUT AREA
ADD A BLANK AFTER OPERAND IN OUTPUT AREA
COMPUTE OPERAND LENGTH (WITH BLANK) & MOVE TO OUTPUT AREA
MOVE SEQUENCE FIELD TO OUTPUT AREA
COMPUTE TOTAL LENGTH & MOVE TO OUTPUT AREA
GLEXC
RETURN TO CALLING PT
E4
COMN
BLANK COMMENT OUTPUT AREA
RIGHT-ADJUST SEQUENCE FIELD IN OUTPUT AREA
XXX
LEFT-ADJUST COMMENT FIELD IN OUTPUT AREA
RDURG RETURN IF INPUT EOF
NOMORE
LOAD FORCED END STATEMENT ADDR AS OUTPUT ADDRESS

CHART CA. IOGET1 ROUTINE

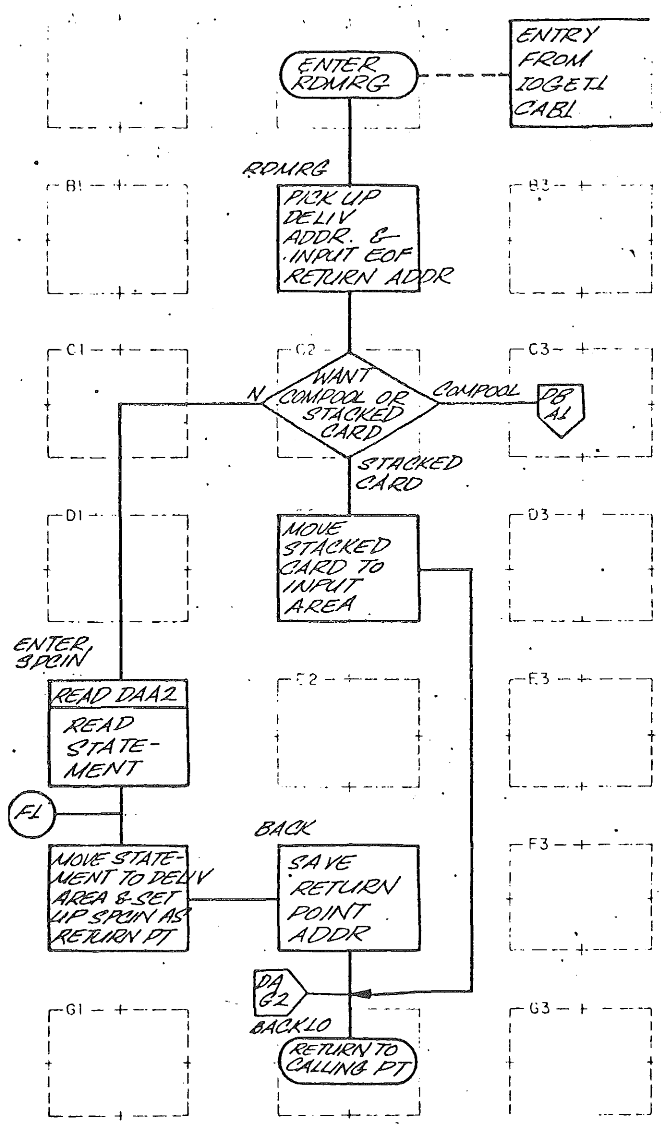


CHART DA. RDMRG, ENTER SUBROUTINES

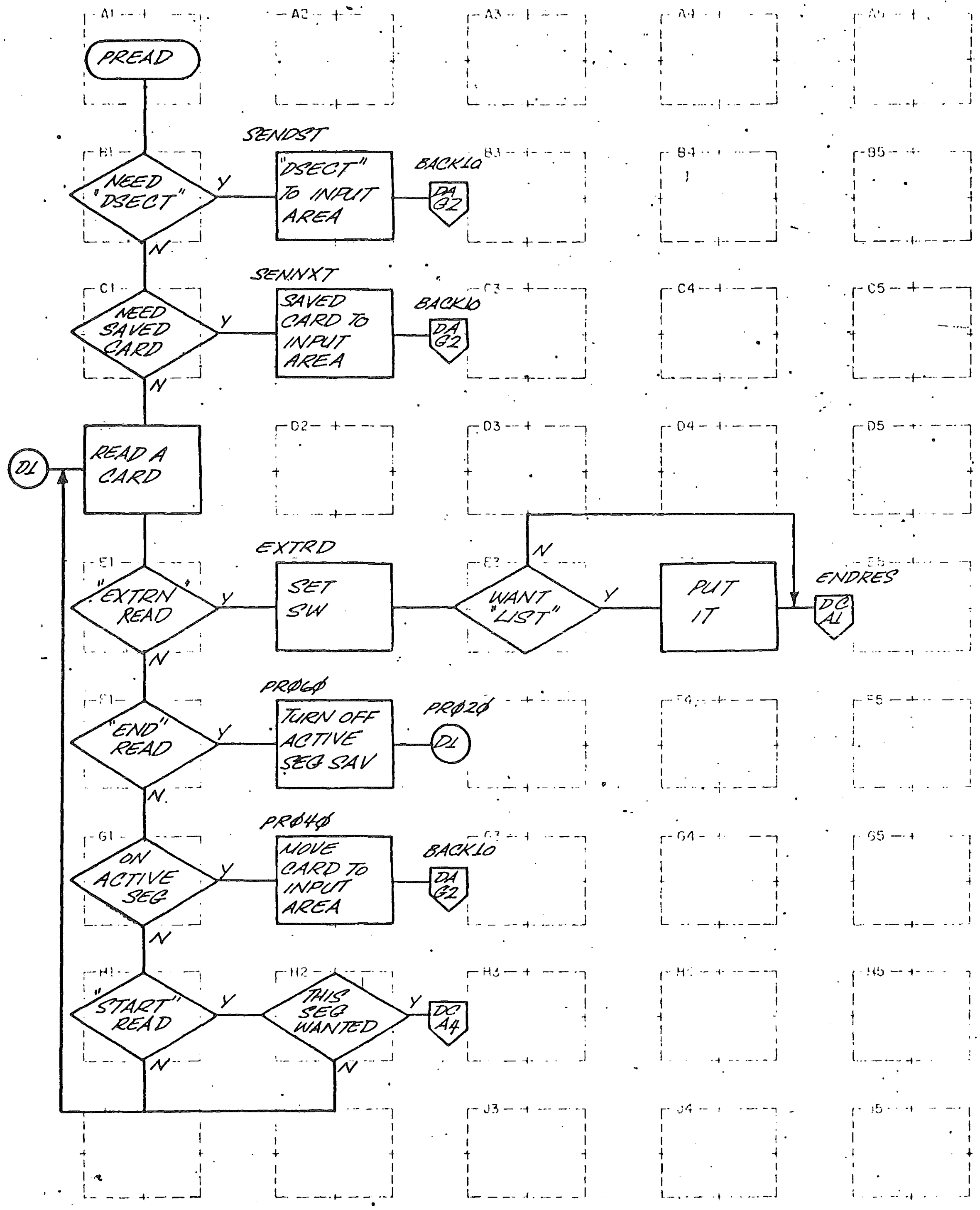


CHART DB. PREAD SUBROUTINE (SHEET 1 OF 2)

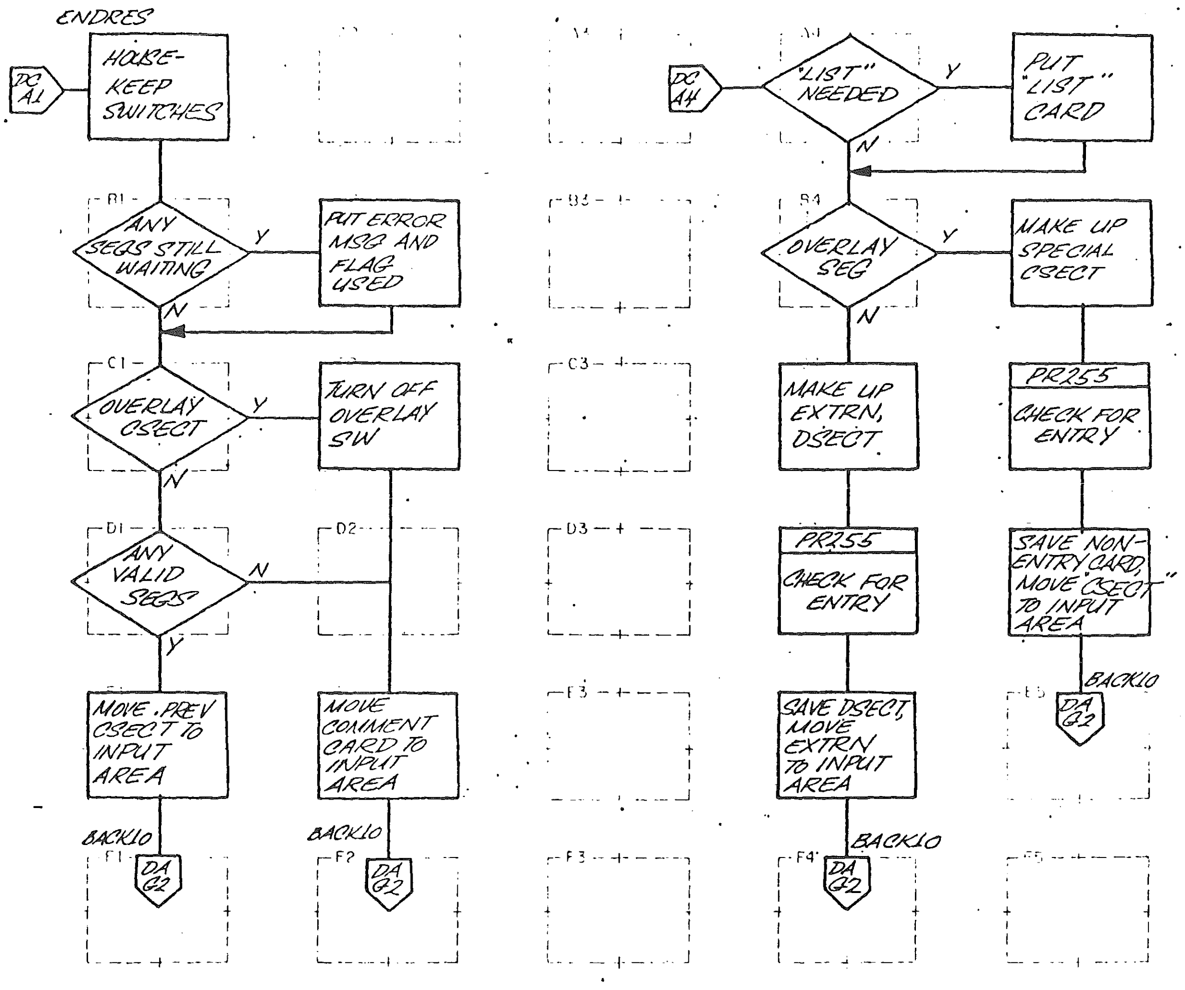


CHART DC. PREAD SUBROUTINE (SHEET 2 OF 2)

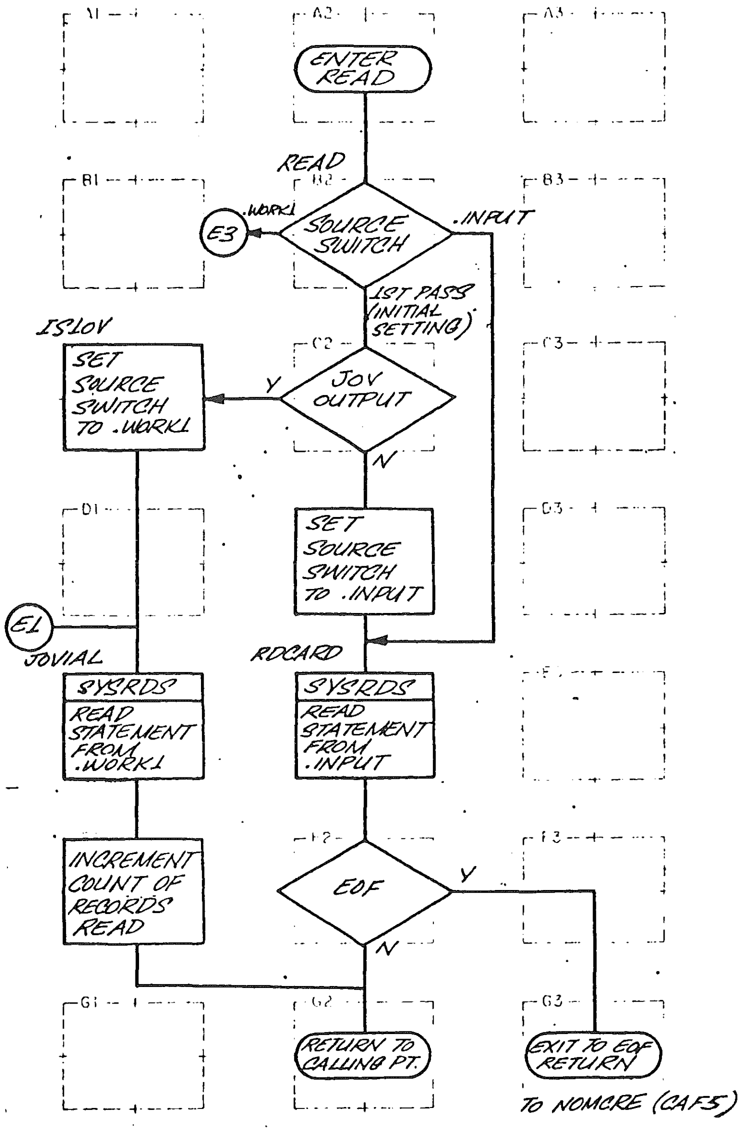
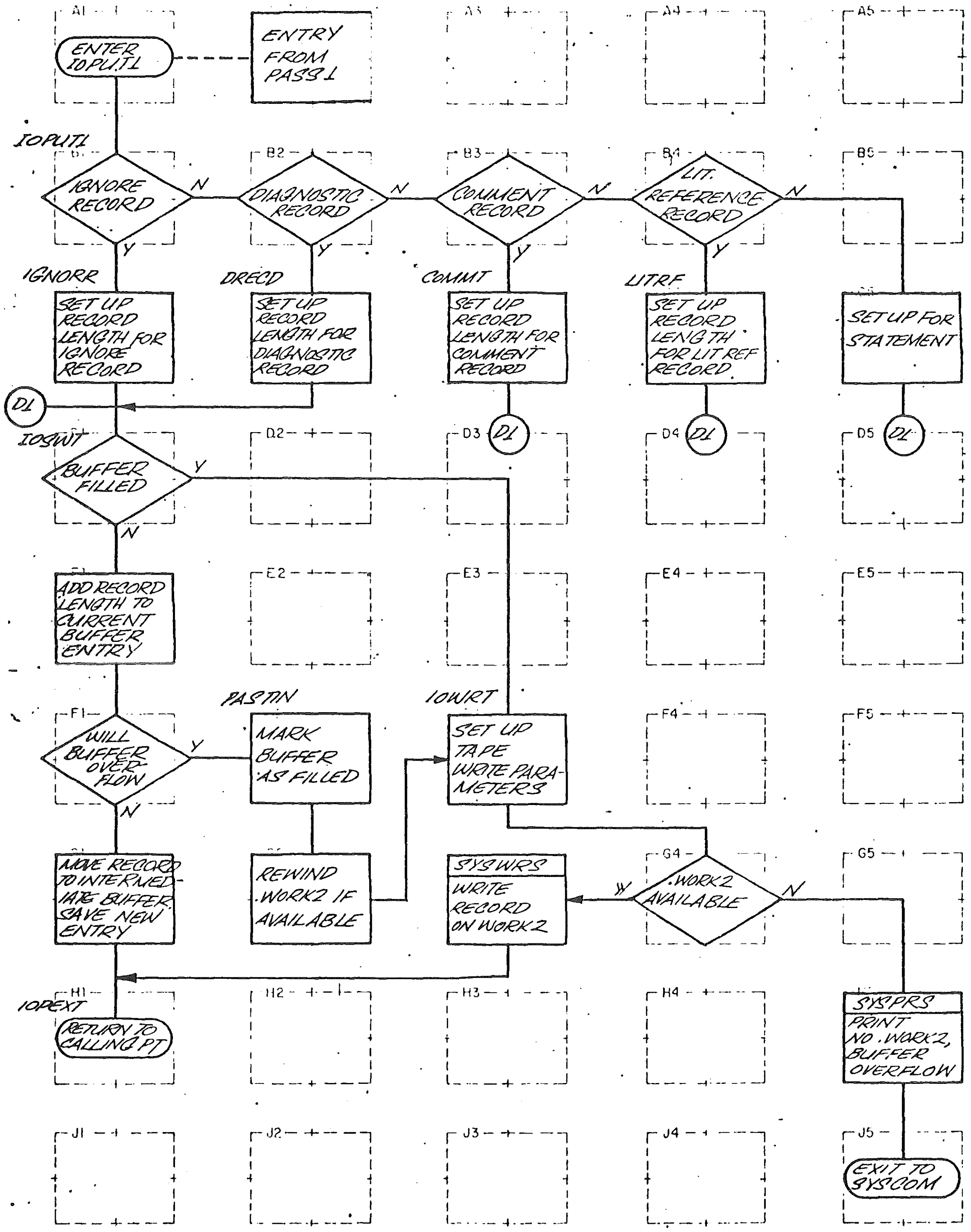

ENTER READ
READ
SOURCE SWITCH
.WORK1
E3
.INPUT
1ST PASS (INITIAL SETTING)
IS1OV
SET SOURCE SWITCH TO .WORK1
Y
JOV OUTPUT
N
SET SOURCE SWITCH TO .INPUT
E1
JOVIAL
SYSRDS
READ STATEMENT FROM .WORK1
RDCARD
SYSRDS
READ STATEMENT FROM .INPUT
INCREMENT COUNT OF RECORDS READ
EOF
Y
N
RETURN TO CALLING PT.
EXIT TO EOF RETURN
TO NOMCRE (CAF5)

CHART DD. READ SUBROUTINE



ENTER IOPUT1
ENTRY FROM PASS 1
IOPUT1
IGNORE RECORD
N
Y
DIAGNOSTIC RECORD
COMMENT RECORD
LIT. REFERENCE RECORD
IGNORR
DRECD
COMMT
LITRF
SET UP RECORD LENGTH FOR IGNORE RECORD
SET UP RECORD LENGTH FOR DIAGNOSTIC RECORD
SET UP RECORD LENGTH FOR COMMENT RECORD
SET UP RECORD LENGTH FOR LIT REF RECORD
SET UP FOR STATEMENT
D1
IOSWT
BUFFER FILLED
ADD RECORD LENGTH TO CURRENT BUFFER ENTRY
PASTIN
IOWRT
WILL BUFFER OVERFLOW
MARK BUFFER AS FILLED
SET UP TAPE WRITE PARAMETERS
MOVE RECORD TO INTERMEDIATE BUFFER. SAVE NEW ENTRY
REWIND .WORK2 IF AVAILABLE
SYSWRS
WRITE RECORD ON WORK2
.WORK2 AVAILABLE
IOPEXT
RETURN TO CALLING PT
SYSPRS
PRINT NO .WORK2, BUFFER OVERFLOW
EXIT TO SYSCOM

CHART NA. IOPUT1 ROUTINE

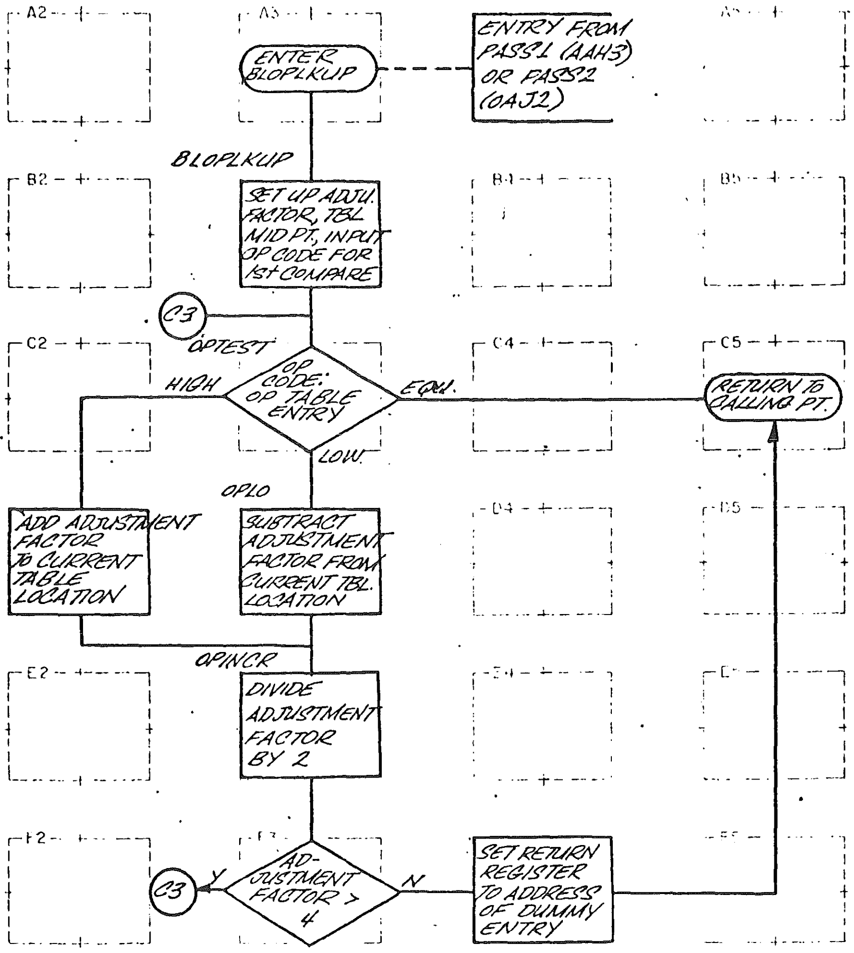

ENTER BLOPLKUP
ENTRY FROM PASS1 (AAH3) OR PASS2 (OAJ2)
BLOPLKUP
SET UP ADJU. FACTOR, TBL MID PT, INPUT OP CODE FOR 1st COMPARE
C3
OPTEST
OP CODE: OP TABLE ENTRY
HIGH
EQU.
LOW
RETURN TO CALLING PT.
OPLO
ADD ADJUSTMENT FACTOR TO CURRENT TABLE LOCATION
SUBTRACT ADJUSTMENT FACTOR FROM CURRENT TBL. LOCATION
OPINCR
DIVIDE ADJUSTMENT FACTOR BY 2
ADJUSTMENT FACTOR > 4
Y
N
C3
SET RETURN REGISTER TO ADDRESS OF DUMMY ENTRY

CHART EA. BLOPLKUP ROUTINE

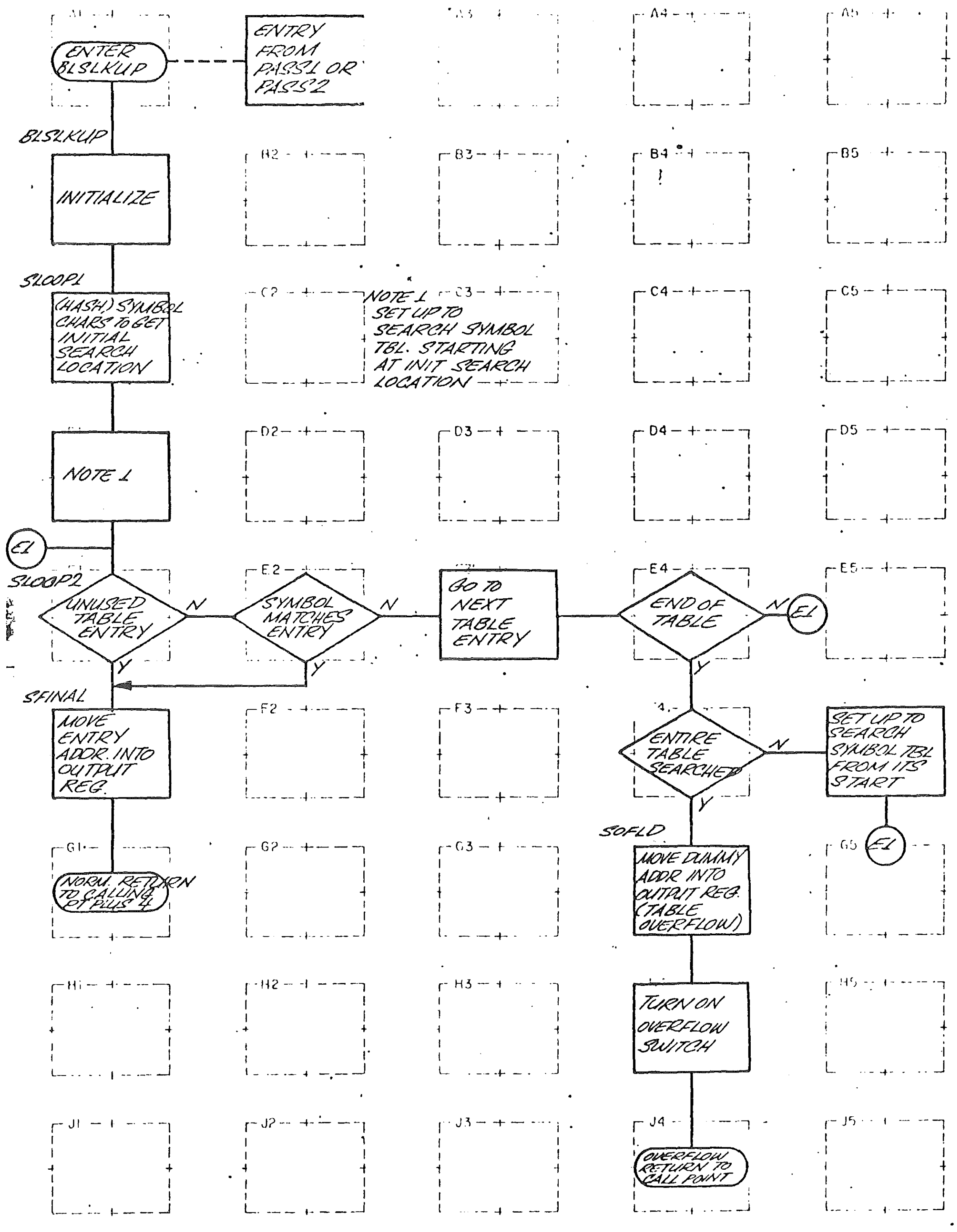
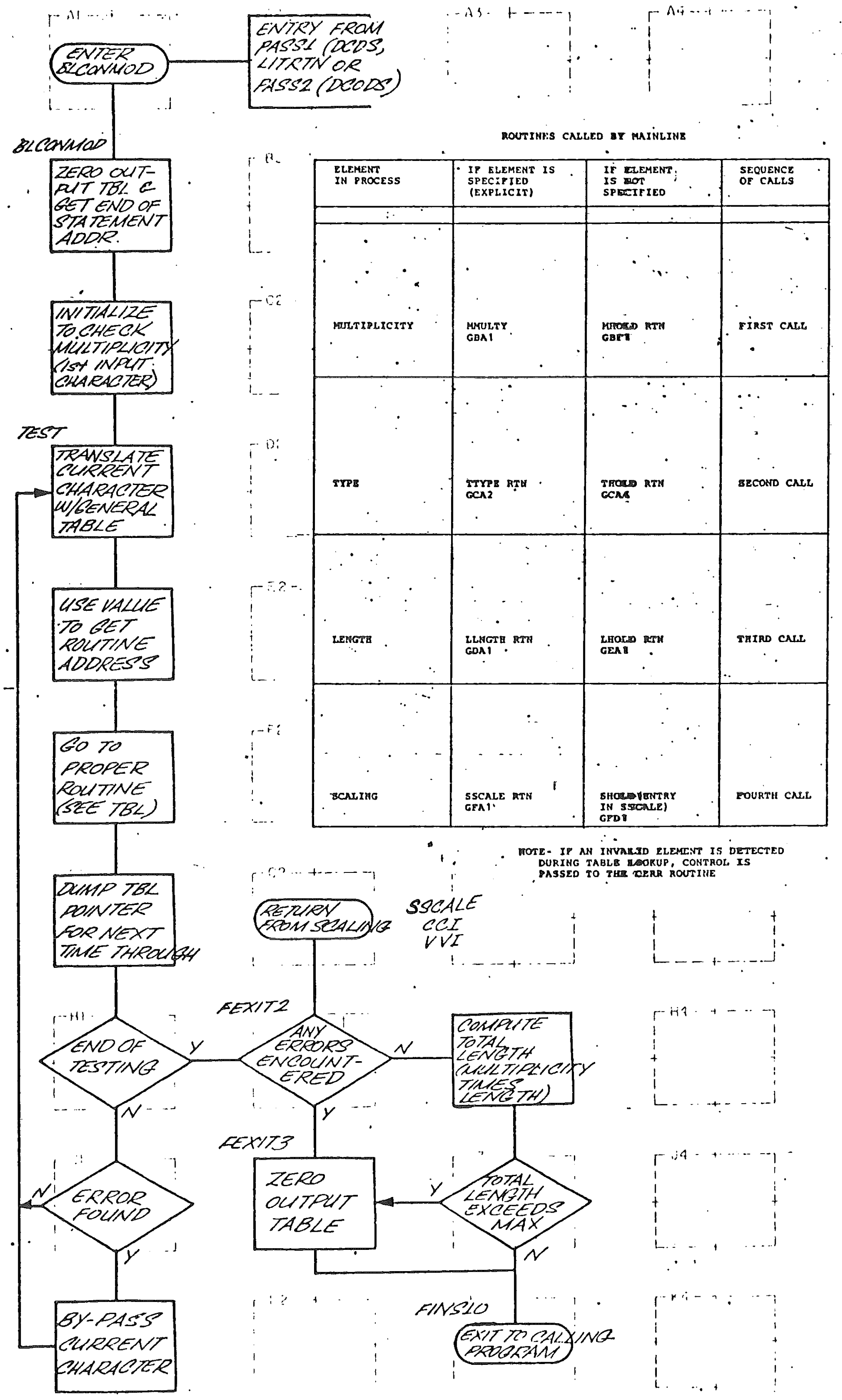

ENTER BLSLKUP
ENTRY FROM PASS1 OR PASS2
BLSLKUP
INITIALIZE
SLOOP1
(HASH) SYMBOL CHARS TO GET INITIAL SEARCH LOCATION
NOTE 1 SET UP TO SEARCH SYMBOL TBL. STARTING AT INIT SEARCH LOCATION
NOTE 1
E1
SLOOP2
UNUSED TABLE ENTRY
N
SYMBOL MATCHES ENTRY
N
GO TO NEXT TABLE ENTRY
END OF TABLE
N
E1
Y
Y
SFINAL
MOVE ENTRY ADDR. INTO OUTPUT REG.
Y
ENTIRE TABLE SEARCHED
N
SET UP TO SEARCH SYMBOL TBL FROM ITS START
E1
Y
SOFLD
NORM. RETURN TO CALLING PT PLUS 4
MOVE DUMMY ADDR INTO OUTPUT REG. (TABLE OVERFLOW)
TURN ON OVERFLOW SWITCH
OVERFLOW RETURN TO CALL POINT

CHART FA. BLSLKUP ROUTINE


ENTER BLCONMOD
ENTRY FROM PASS1 (DCDS, LITRTN OR PASS2 (DCODS)
BLCONMOD
ZERO OUTPUT TBL & GET END OF STATEMENT ADDR.
INITIALIZE TO CHECK MULTIPLICITY (1st INPUT CHARACTER)
TEST
TRANSLATE CURRENT CHARACTER W/GENERAL TABLE
USE VALUE TO GET ROUTINE ADDRESS
GO TO PROPER ROUTINE (SEE TBL)
DUMP TBL POINTER FOR NEXT TIME THROUGH
END OF TESTING
Y
N
ERROR FOUND
N
Y
BY-PASS CURRENT CHARACTER
RETURN FROM SCALING
SSCALE
CCI
VVI
FEXIT2
ANY ERRORS ENCOUNTERED
N
Y
COMPUTE TOTAL LENGTH (MULTIPLICITY TIMES LENGTH)
FEXIT3
ZERO OUTPUT TABLE
TOTAL LENGTH EXCEEDS MAX
Y
N
FINSIO
EXIT TO CALLING PROGRAM

ROUTINES CALLED BY MAINLINE

ELEMENT IN PROCESS	IF ELEMENT IS SPECIFIED (EXPLICIT)	IF ELEMENT IS NOT SPECIFIED	SEQUENCE OF CALLS
MULTIPLICITY	MMULTY GBA1	MHOLD RTN GBF1	FIRST CALL
TYPE	TTYPE RTN GCA2	THOLD RTN GCA4	SECOND CALL
LENGTH	LLNGTH RTN GDA1	LHOLD RTN GEA1	THIRD CALL
SCALING	SSCALE RTN GFA1	SHOLD (ENTRY IN SSCALE) GFD1	FOURTH CALL

NOTE- IF AN INVALID ELEMENT IS DETECTED DURING TABLE LOOKUP, CONTROL IS PASSED TO THE CERR ROUTINE

CHART GA. BLCONMOD MAINLINE

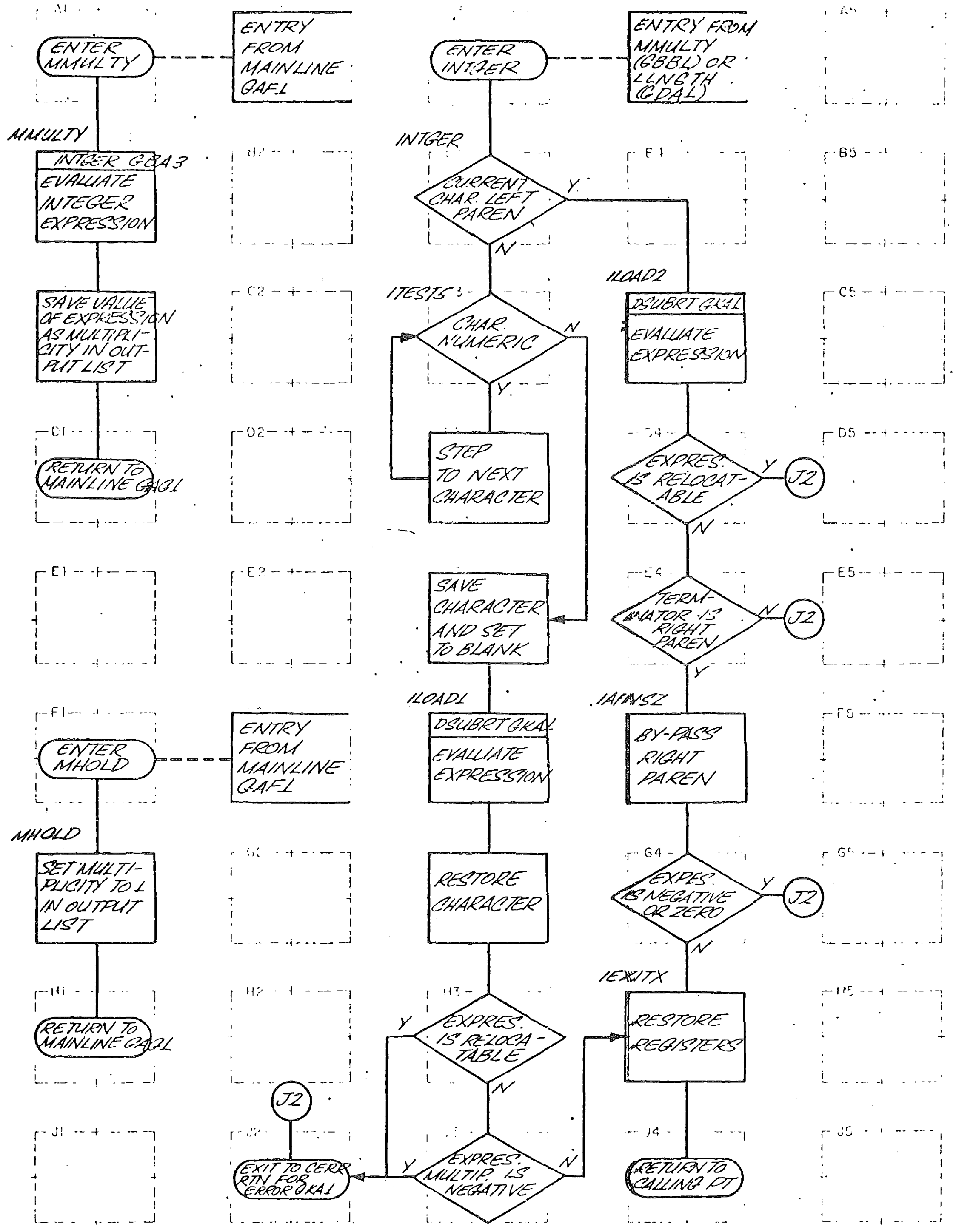


CHART GB. MMULTY, MHOLD ROUTINES, INTGER SUBROUTINE

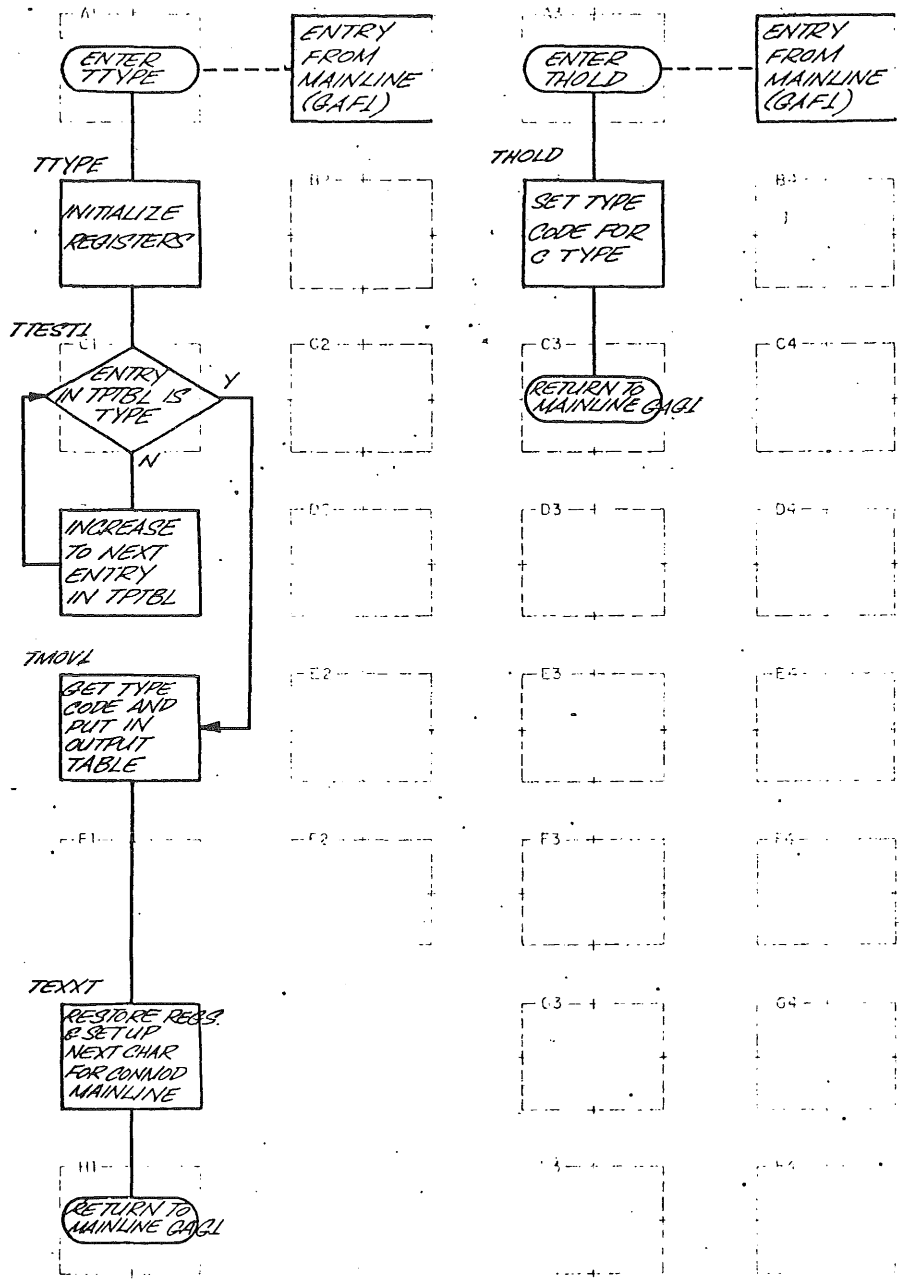

ENTER TTYPE
ENTRY FROM MAINLINE (GAF1)
TTYPE
INITIALIZE REGISTERS
TTEST1
ENTRY IN TPTBL IS TYPE
Y
N
INCREASE TO NEXT ENTRY IN TPTBL
TMOV1
GET TYPE CODE AND PUT IN OUTPUT TABLE
TEXXT
RESTORE REGS. & SET UP NEXT CHAR FOR COMMOD MAINLINE
RETURN TO MAINLINE GAG1
ENTER THOLD
ENTRY FROM MAINLINE (GAF1)
THOLD
SET TYPE CODE FOR C TYPE
RETURN TO MAINLINE GAG1

CHART GC. TTYPE, THOLD ROUTINES

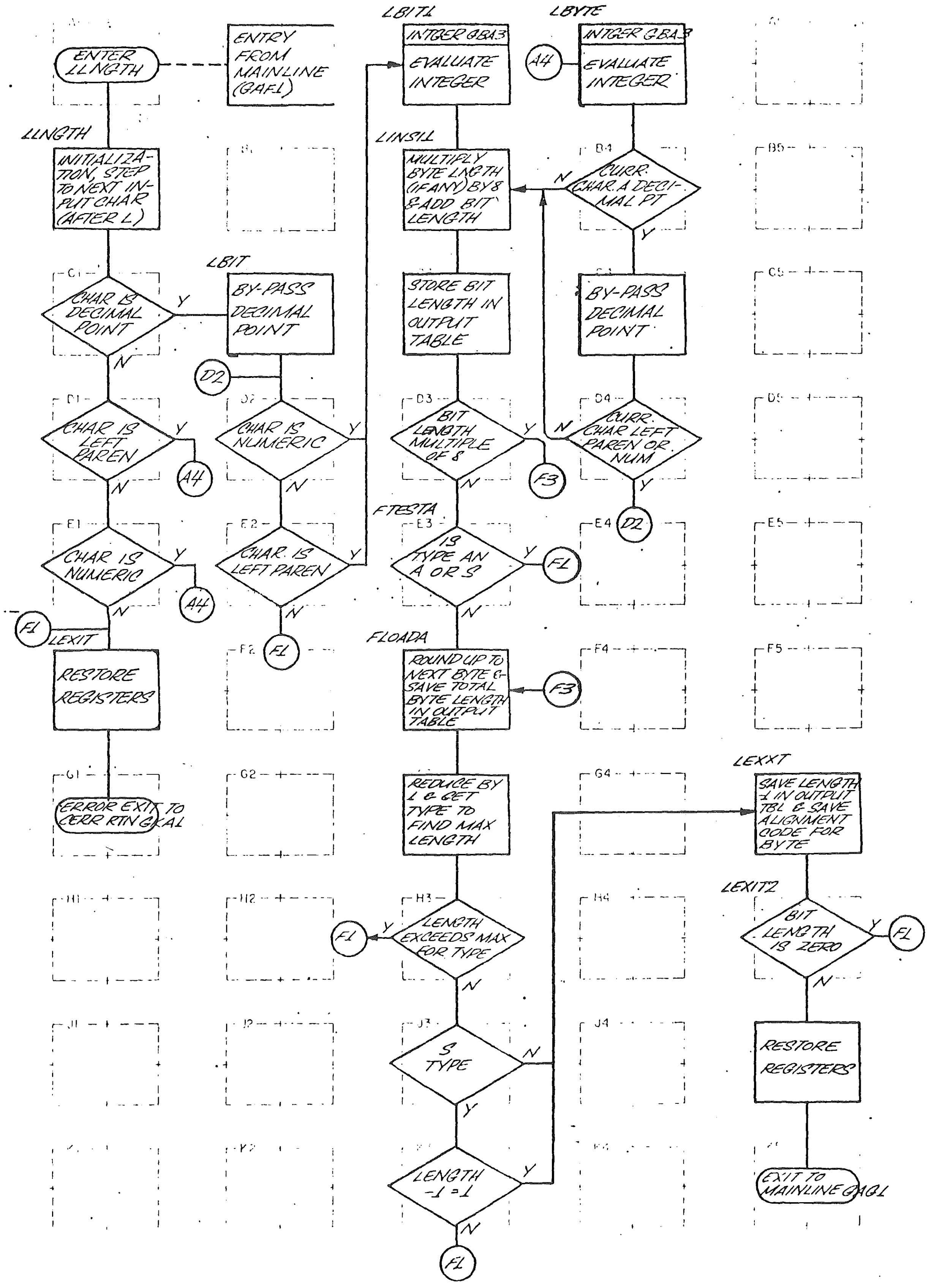


CHART GD. LLNGTH ROUTINE

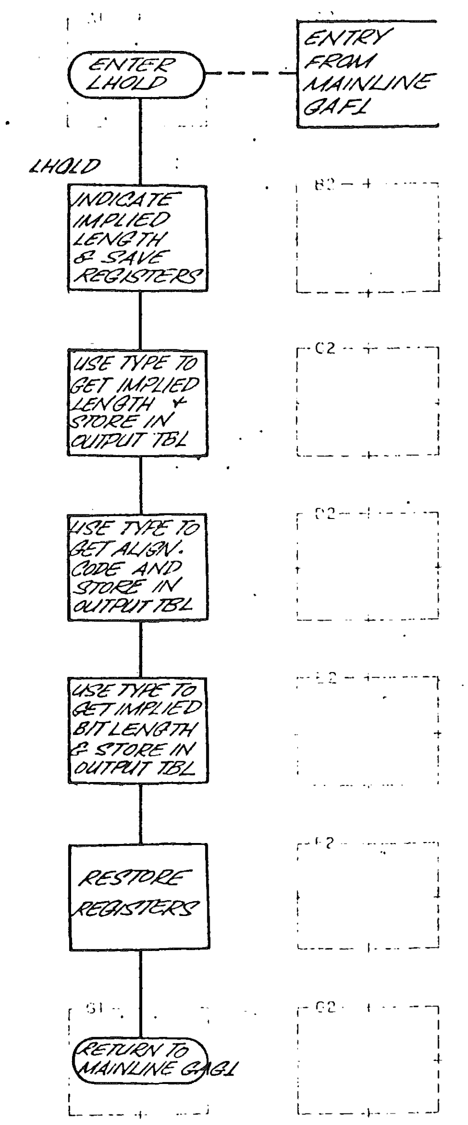

ENTER LHOLD
ENTRY FROM MAINLINE GAF1
LHOLD
INDICATE IMPLIED LENGTH & SAVE REGISTERS
USE TYPE TO GET IMPLIED LENGTH + STORE IN OUTPUT TBL
USE TYPE TO GET ALIGN. CODE AND STORE IN OUTPUT TBL
USE TYPE TO GET IMPLIED BIT LENGTH & STORE IN OUTPUT TBL
RESTORE REGISTERS
RETURN TO MAINLINE GAG1

CHART GE. LHOLD ROUTINE

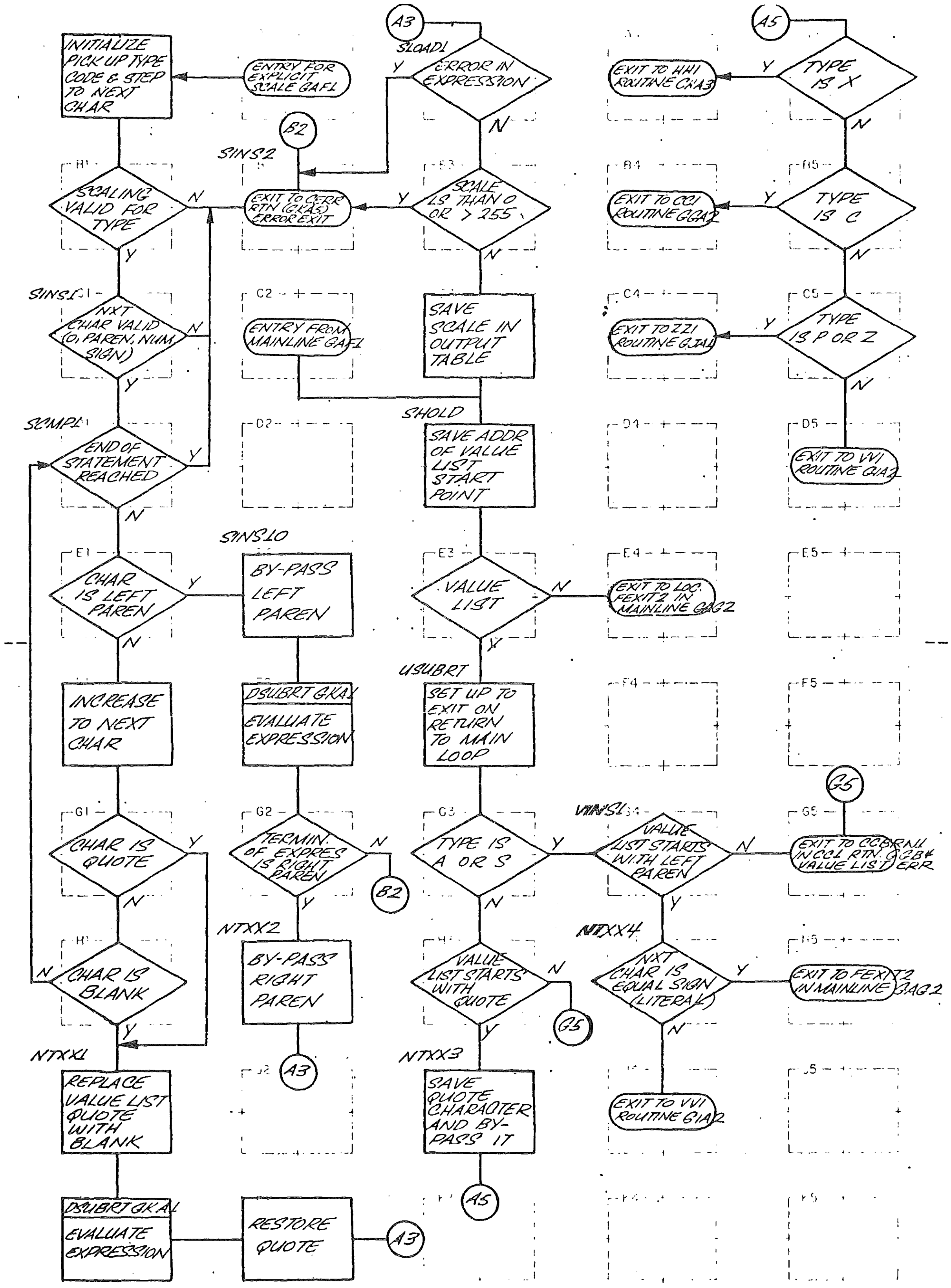

INITIALIZE PICK UP TYPE CODE & STEP TO NEXT CHAR
ENTRY FOR EXPLICIT SCALE GAF1
A3
SLOAD1
ERROR IN EXPRESSION
A5
TYPE IS X
EXIT TO HH1 ROUTINE GHA3
B2
SINS2
SCALING VALID FOR TYPE
EXIT TO CERR RTN (GKAS) ERROR EXIT
SCALE LS THAN 0 OR > 255
TYPE IS C
EXIT TO CC1 ROUTINE GGA2
SINS1
NXT CHAR VALID (0, PAREN, NUM, SIGN)
ENTRY FROM MAINLINE GAF1
SAVE SCALE IN OUTPUT TABLE
TYPE IS P OR Z
EXIT TO ZZ1 ROUTINE GJA1
SCMPL
END OF STATEMENT REACHED
SHOLD
SAVE ADDR OF VALUE LIST START POINT
EXIT TO VV1 ROUTINE GIA2
SINS10
CHAR IS LEFT PAREN
BY-PASS LEFT PAREN
VALUE LIST
EXIT TO LOC. FEXIT2 IN MAINLINE GAG2
INCREASE TO NEXT CHAR
DSUBRT GKA1
EVALUATE EXPRESSION
USUBRT
SET UP TO EXIT ON RETURN TO MAIN LOOP
G5
CHAR IS QUOTE
TERMIN OF EXPRES IS RIGHT PAREN
TYPE IS A OR S
VNNS1
VALUE LIST STARTS WITH LEFT PAREN
EXIT TO CCBRN1 IN CC1 RTN. GCB4 VALUE LIST ERR
NTXX2
NTXX4
CHAR IS BLANK
BY-PASS RIGHT PAREN
VALUE LIST STARTS WITH QUOTE
NXT CHAR IS EQUAL SIGN (LITERAL)
EXIT TO FEXIT2 IN MAINLINE GAG2
NTXX1
NTXX3
REPLACE VALUE LIST QUOTE WITH BLANK
SAVE QUOTE CHARACTER AND BY-PASS IT
EXIT TO VV1 ROUTINE GIA2
DSUBRT GKA1
EVALUATE EXPRESSION
RESTORE QUOTE
A5
A3
Y
N

CHART GF. SSCALE ROUTINE

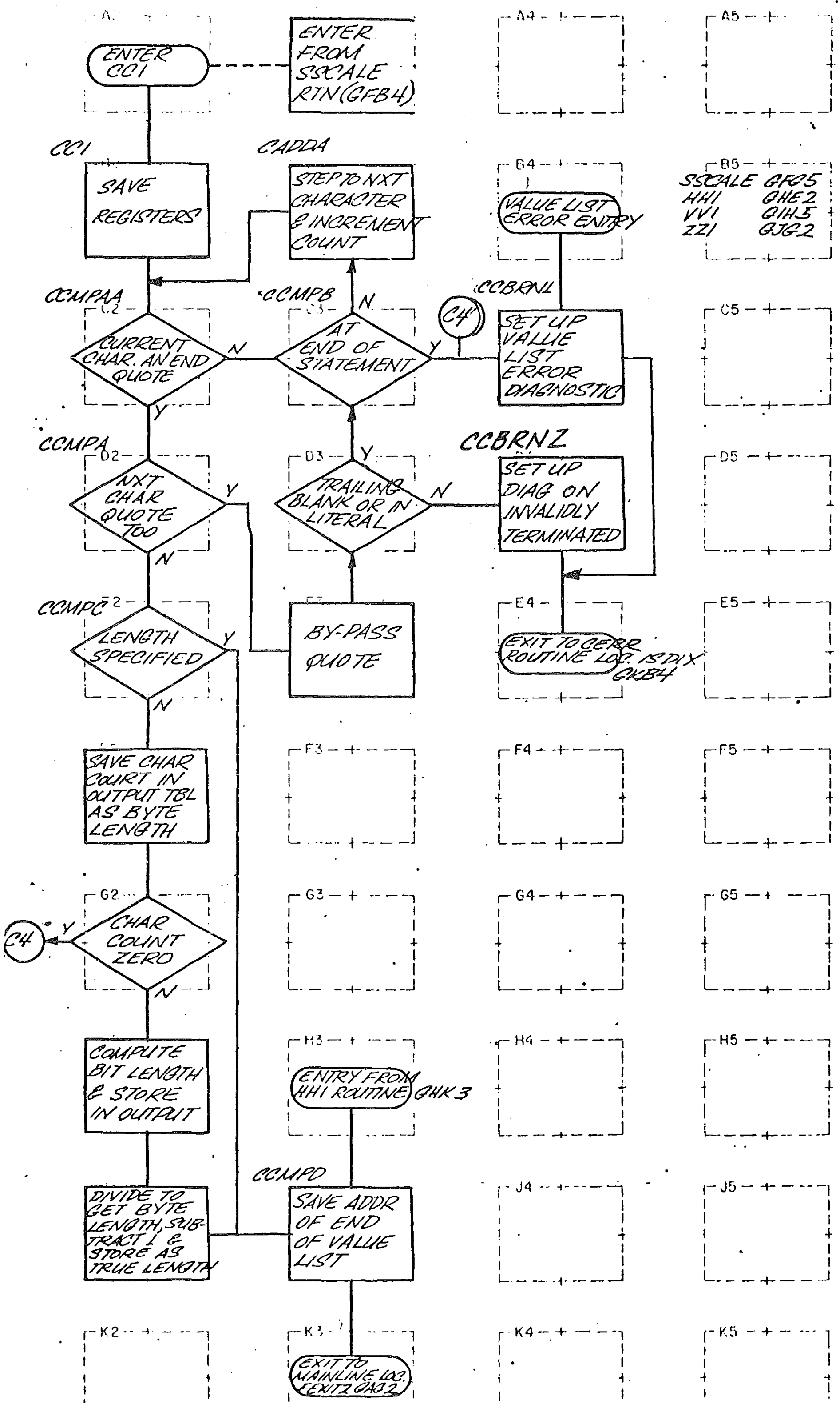
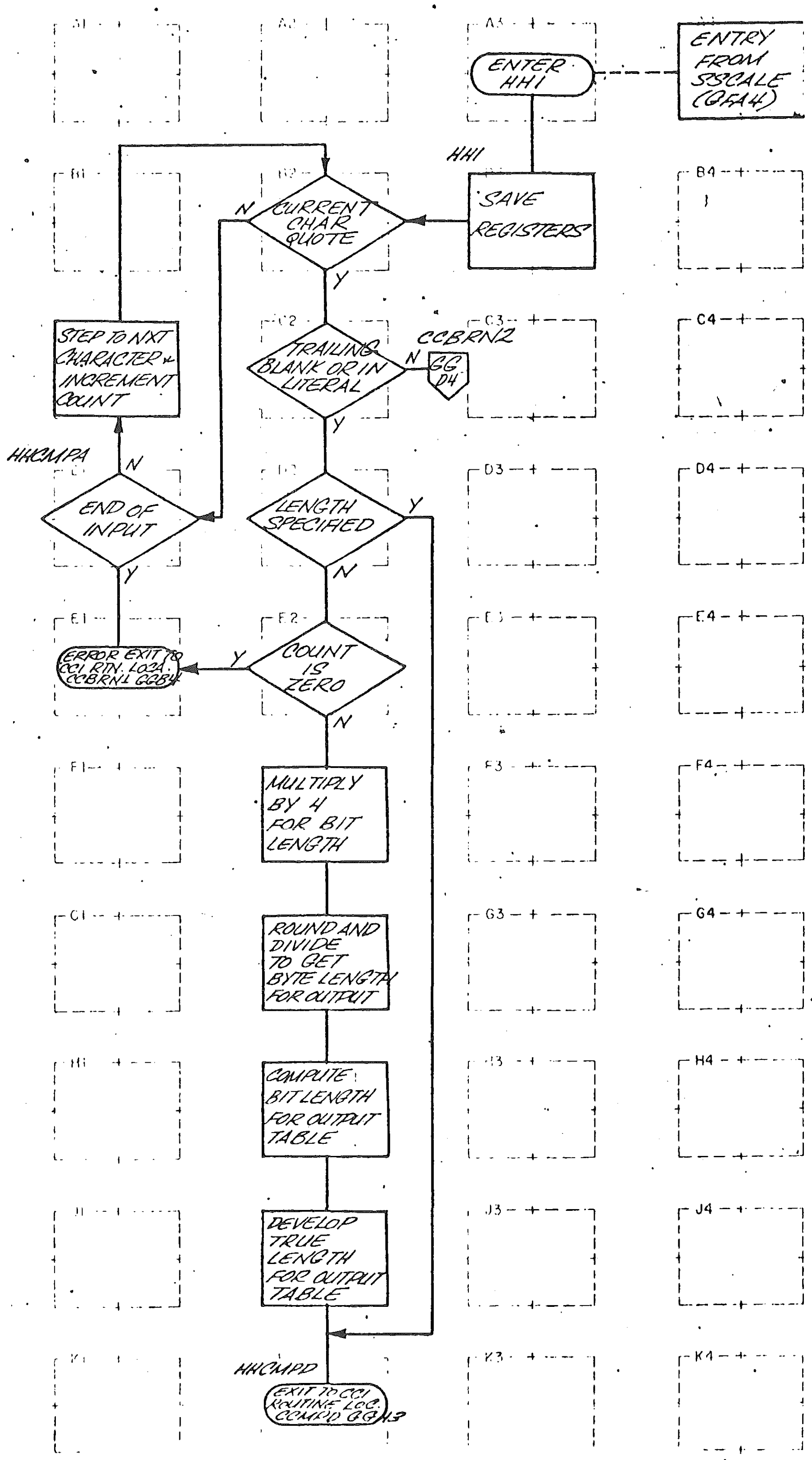

ENTER CCI
ENTER FROM SSCALE RTN(GFB4)
CCI
SAVE REGISTERS
CADDA
STEP TO NXT CHARACTER & INCREMENT COUNT
VALUE LIST ERROR ENTRY
SSCALE GFG5
HHI GHE2
VVI GIH5
ZZI GJG2
CCMPAA
CURRENT CHAR. AN END QUOTE
CCMPB
AT END OF STATEMENT
CCBRNL
C4
SET UP VALUE LIST ERROR DIAGNOSTIC
CCMPA
NXT CHAR QUOTE TOO
TRAILING BLANK OR IN LITERAL
CCBRNZ
SET UP DIAG ON INVALIDLY TERMINATED
CCMPC
LENGTH SPECIFIED
BY-PASS QUOTE
EXIT TO CERR ROUTINE LOC. ISDIX GKB4
SAVE CHAR COUNT IN OUTPUT TBL AS BYTE LENGTH
CHAR COUNT ZERO
C4
COMPUTE BIT LENGTH & STORE IN OUTPUT
ENTRY FROM HHI ROUTINE GHK3
CCMPD
DIVIDE TO GET BYTE LENGTH, SUBTRACT 1 & STORE AS TRUE LENGTH
SAVE ADDR OF END OF VALUE LIST
EXIT TO MAINLINE LOC. FEXIT2 GAB2

CHART GG. CCI ROUTINE



ENTER HHI
ENTRY FROM SSSCALE (GFA4)
HHI
SAVE REGISTERS
CURRENT CHAR QUOTE
N
Y
TRAILING BLANK OR IN LITERAL
CCBRN2
GG D4
LENGTH SPECIFIED
COUNT IS ZERO
STEP TO NXT CHARACTER & INCREMENT COUNT
HHCMPA
END OF INPUT
ERROR EXIT TO CCI RTN. LOC. CCBRN1 GGB4
MULTIPLY BY 4 FOR BIT LENGTH
ROUND AND DIVIDE TO GET BYTE LENGTH FOR OUTPUT
COMPUTE BIT LENGTH FOR OUTPUT TABLE
DEVELOP TRUE LENGTH FOR OUTPUT TABLE
HHCMPD
EXIT TO CCI ROUTINE LOC. CCMPD GGH3

CHART GH. HHI ROUTINE

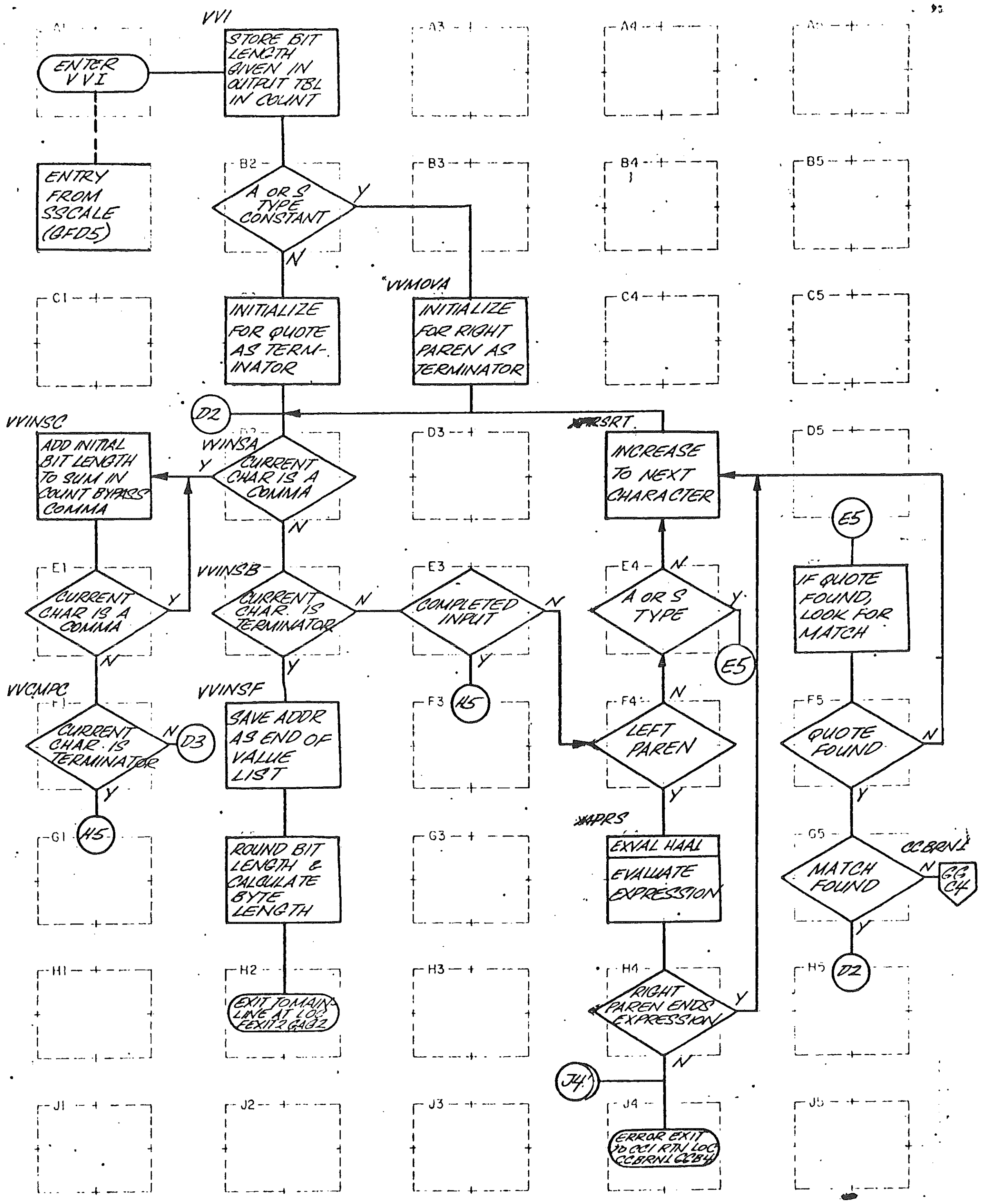

ENTER VVI
ENTRY FROM SSCALE (GFD5)
VVI
STORE BIT LENGTH GIVEN IN OUTPUT TBL IN COUNT
A OR S TYPE CONSTANT
Y
N
INITIALIZE FOR QUOTE AS TERMINATOR
VVMOVA
INITIALIZE FOR RIGHT PAREN AS TERMINATOR
D2
SRT
INCREASE TO NEXT CHARACTER
VVINSC
ADD INITIAL BIT LENGTH TO SUM IN COUNT BYPASS COMMA
VVINSA
CURRENT CHAR IS A COMMA
CURRENT CHAR IS A COMMA
VVINSB
CURRENT CHAR IS TERMINATOR
COMPLETED INPUT
A OR S TYPE
E5
IF QUOTE FOUND, LOOK FOR MATCH
VVCMPC
CURRENT CHAR IS TERMINATOR
D3
VVINSF
SAVE ADDR AS END OF VALUE LIST
H5
LEFT PAREN
QUOTE FOUND
APRS
EXVAL HAAL
EVALUATE EXPRESSION
CCBRNL
MATCH FOUND
GG C4
ROUND BIT LENGTH & CALCULATE BYTE LENGTH
EXIT TO MAINLINE AT LOC FEXIT2 GAG2
RIGHT PAREN ENDS EXPRESSION
J4
ERROR EXIT TO CCI RTN LOC CCBRNL CCB4

CHART GI. VVI ROUTINE

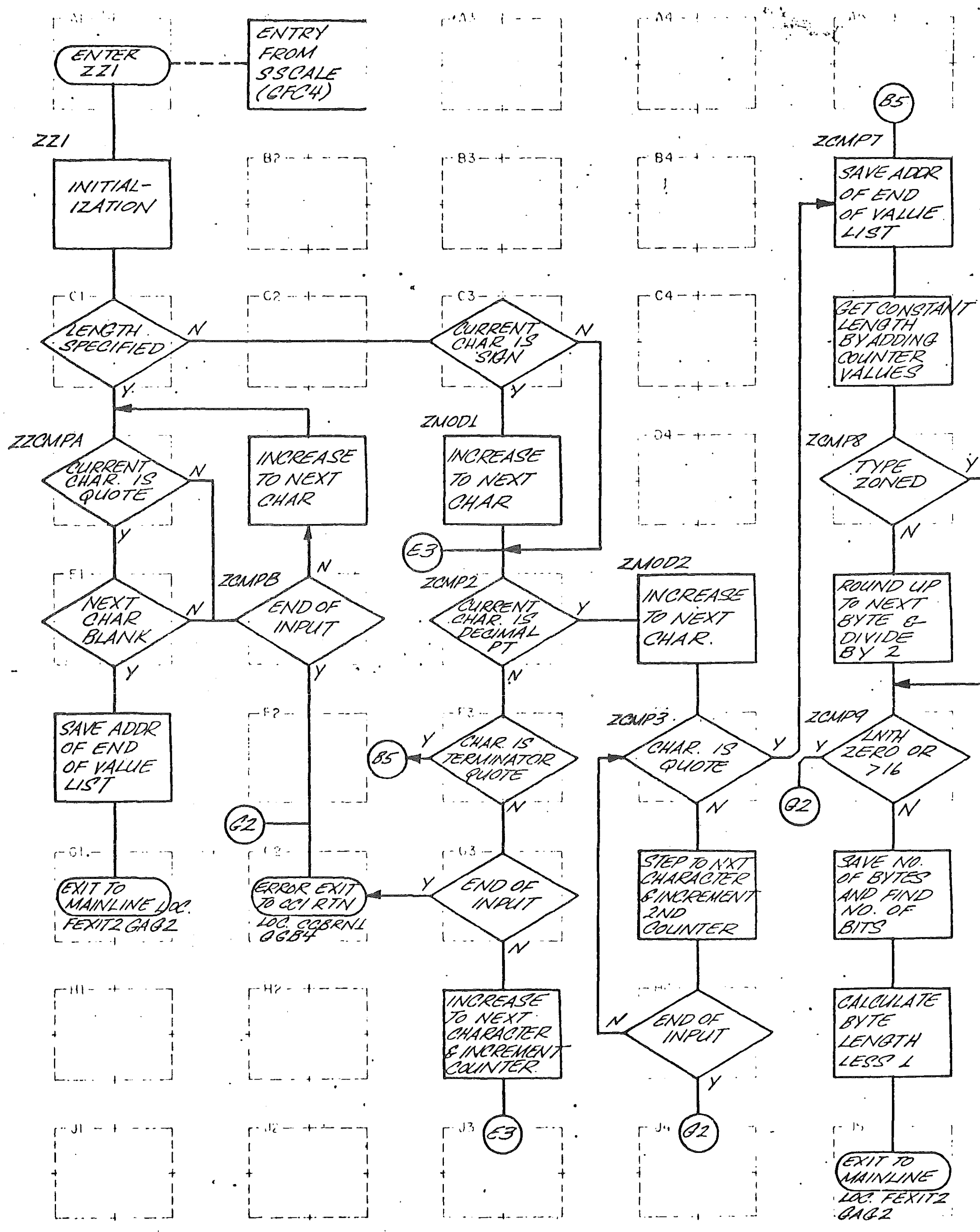

ENTER ZZI
ENTRY FROM SSCALE (CFC4)
ZZI
INITIAL-IZATION
LENGTH SPECIFIED
N
Y
CURRENT CHAR IS SIGN
ZZCMPA
CURRENT CHAR. IS QUOTE
INCREASE TO NEXT CHAR
ZMODL
INCREASE TO NEXT CHAR
E3
ZCMPB
NEXT CHAR BLANK
END OF INPUT
ZCMP2
CURRENT CHAR. IS DECIMAL PT
ZMOD2
INCREASE TO NEXT CHAR.
SAVE ADDR OF END OF VALUE LIST
G2
B5
CHAR. IS TERMINATOR QUOTE
ZCMP3
CHAR. IS QUOTE
EXIT TO MAINLINE LOC. FEXIT2 GAG2
ERROR EXIT TO CCI RTN LOC. CC8RNL 0GB4
END OF INPUT
STEP TO NXT CHARACTER & INCREMENT 2ND COUNTER
INCREASE TO NEXT CHARACTER & INCREMENT COUNTER
END OF INPUT
E3
G2
ZCMP7
SAVE ADDR OF END OF VALUE LIST
GET CONSTANT LENGTH BY ADDING COUNTER VALUES
ZCMP8
TYPE ZONED
ROUND UP TO NEXT BYTE & DIVIDE BY 2
ZCMP9
LNTH ZERO OR 716
SAVE NO. OF BYTES AND FIND NO. OF BITS
CALCULATE BYTE LENGTH LESS 1
EXIT TO MAINLINE LOC. FEXIT2 GAG2

CHART GJ. ZZI ROUTINE

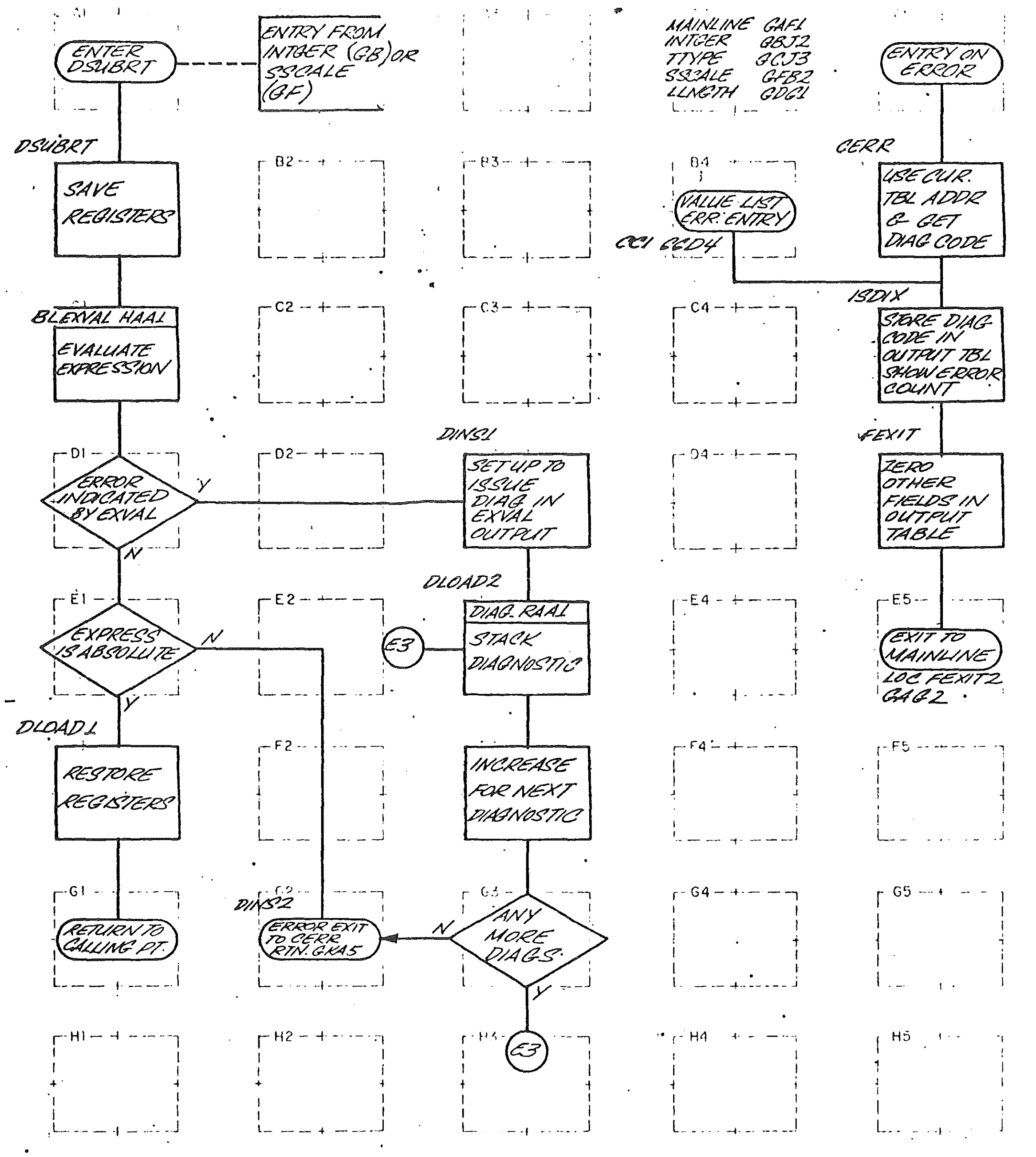


CHART GK. DSUBRT SUBROUTINE, CERR ROUTINE

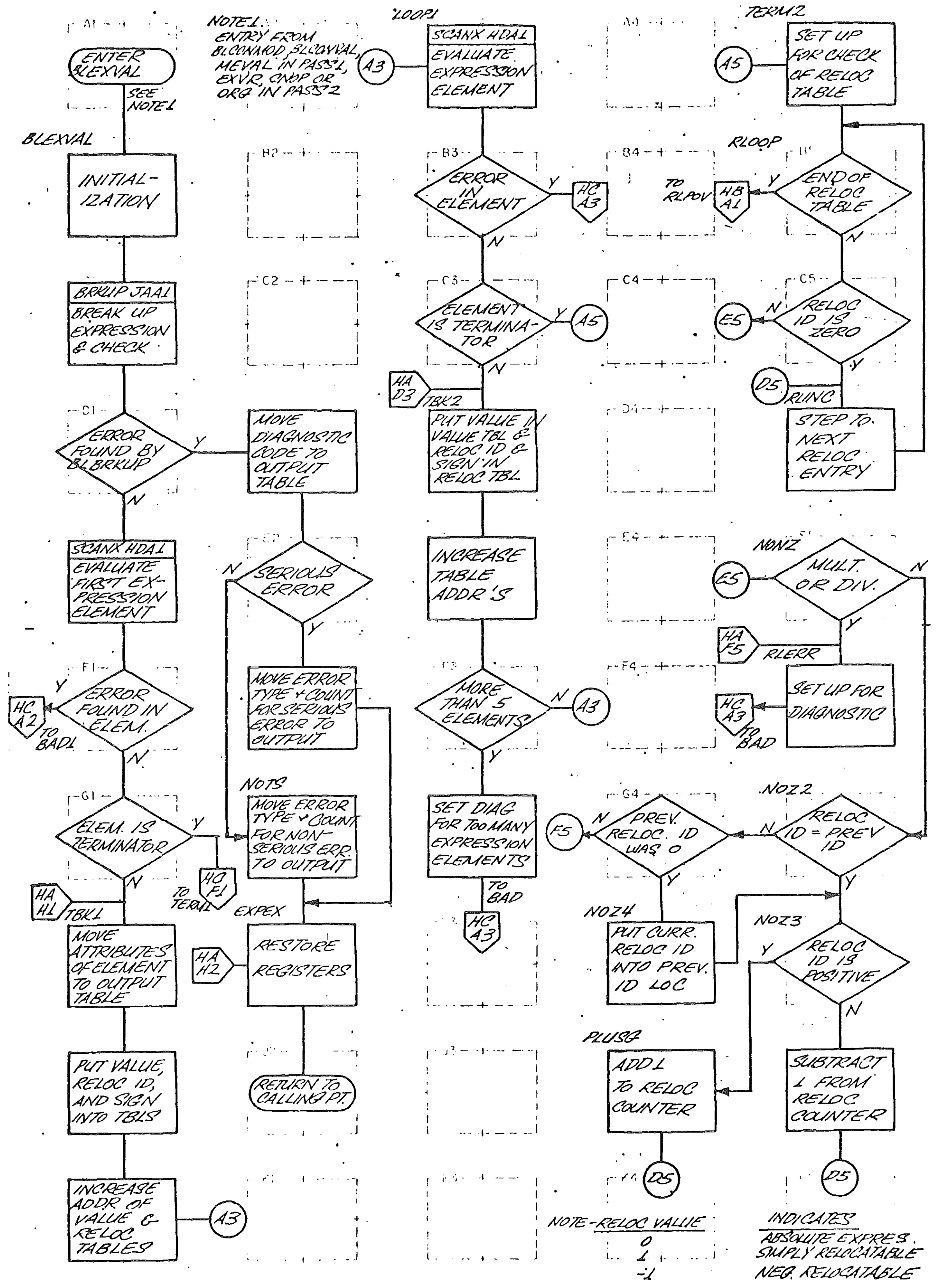


CHART HA. EXVAL MAINLINE (SHEET 1 OF 3)

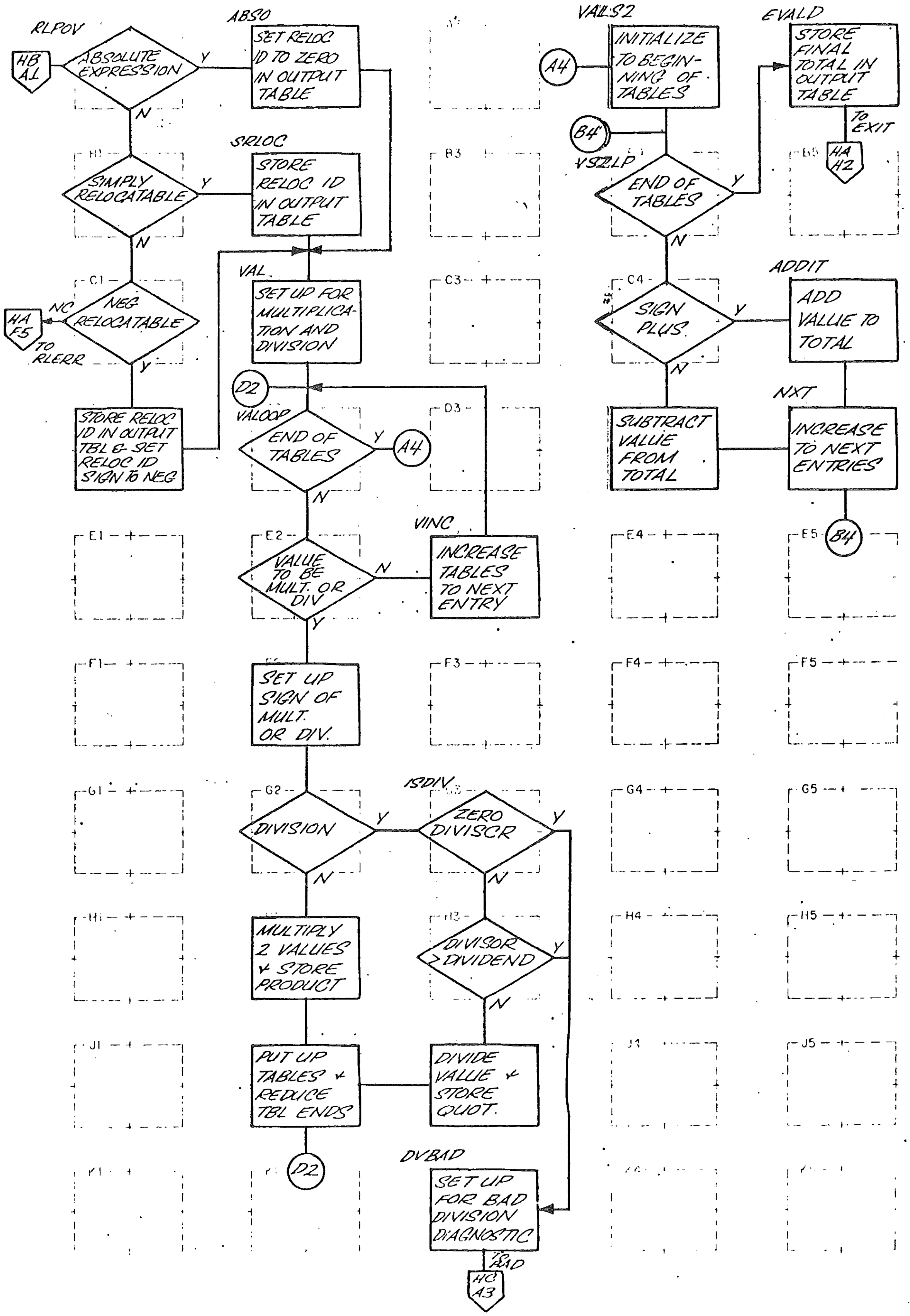

RLPOV
HB A1
ABSOLUTE EXPRESSION
Y
N
ABSO
SET RELOC ID TO ZERO IN OUTPUT TABLE
SRLOC
SIMPLY RELOCATABLE
STORE RELOC ID IN OUTPUT TABLE
NEG RELOCATABLE
NC
HA F5
TO RLERR
STORE RELOC ID IN OUTPUT TBL & SET RELOC ID SIGN TO NEG
VAL
SET UP FOR MULTIPLICATION AND DIVISION
D2
VALOOP
END OF TABLES
A4
VALUE TO BE MULT. OR DIV
VINC
INCREASE TABLES TO NEXT ENTRY
SET UP SIGN OF MULT. OR DIV.
DIVISION
ISDIV
ZERO DIVISOR
DIVISOR > DIVIDEND
MULTIPLY 2 VALUES & STORE PRODUCT
PUT UP TABLES & REDUCE TBL ENDS
DIVIDE VALUE & STORE QUOT.
DVBAD
SET UP FOR BAD DIVISION DIAGNOSTIC
TO BAD
HC A3
VALS2
INITIALIZE TO BEGINNING OF TABLES
B4
VS2LP
END OF TABLES
EVALD
STORE FINAL TOTAL IN OUTPUT TABLE
TO EXIT
HA H2
SIGN PLUS
ADDIT
ADD VALUE TO TOTAL
SUBTRACT VALUE FROM TOTAL
NXT
INCREASE TO NEXT ENTRIES
B4

CHART HB. EXVAL MAINLINE (SHEET 2 OF 3)

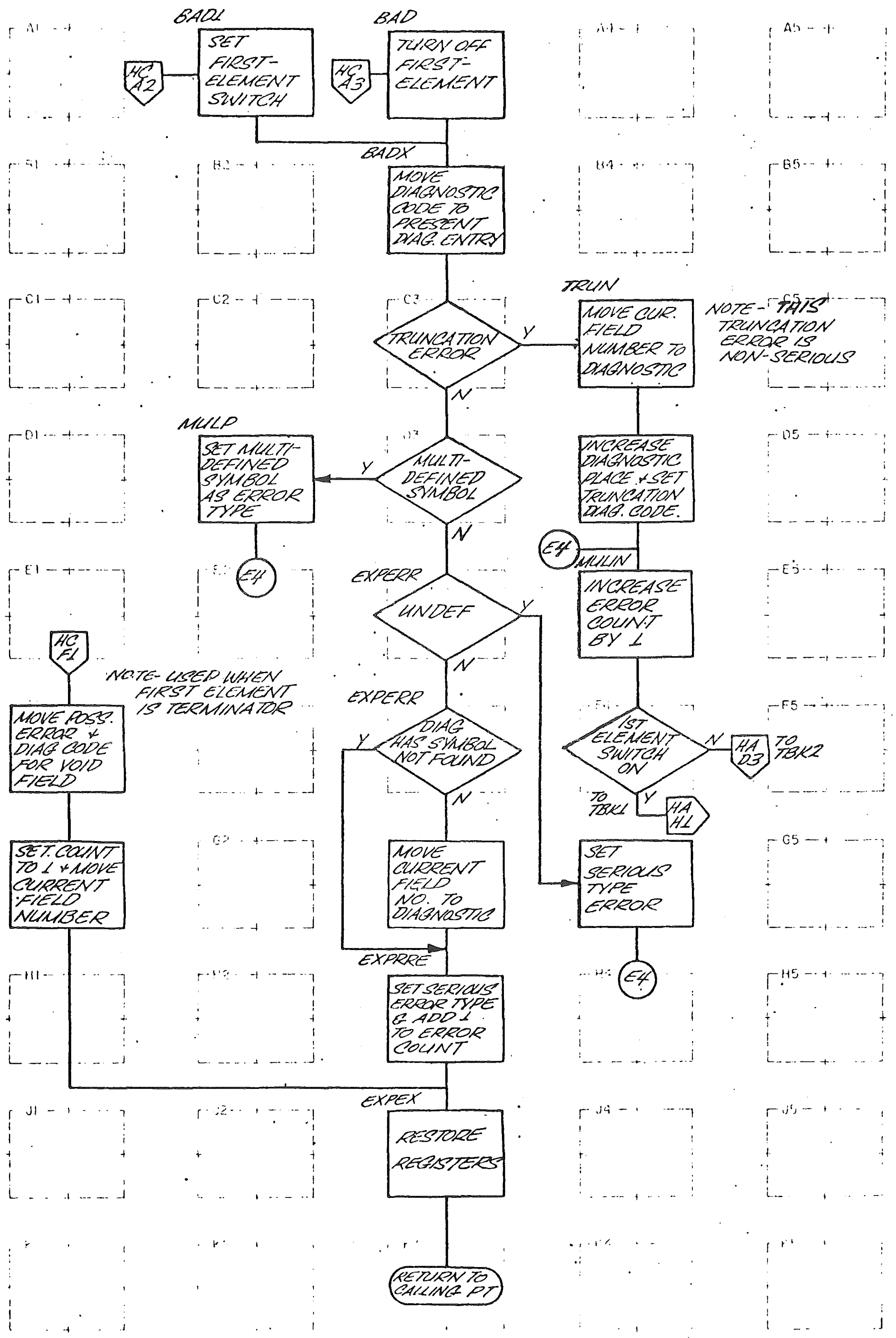

BADL
SET FIRST-ELEMENT SWITCH
HC A2
BAD
TURN OFF FIRST-ELEMENT
HC A3
BADX
MOVE DIAGNOSTIC CODE TO PRESENT DIAG. ENTRY
TRUNCATION ERROR
Y
N
TRUN
MOVE CUR. FIELD NUMBER TO DIAGNOSTIC
NOTE - THIS TRUNCATION ERROR IS NON-SERIOUS
MULP
SET MULTI-DEFINED SYMBOL AS ERROR TYPE
MULTI-DEFINED SYMBOL
INCREASE DIAGNOSTIC PLACE + SET TRUNCATION DIAG. CODE.
E4
MULN
EXPERR
UNDEF
INCREASE ERROR COUNT BY 1
HC F1
NOTE- USED WHEN FIRST ELEMENT IS TERMINATOR
MOVE POSS. ERROR + DIAG CODE FOR VOID FIELD
EXPERR
DIAG HAS SYMBOL NOT FOUND
1ST ELEMENT SWITCH ON
HA D3
TO TBK2
TO TBK1
HA H1
SET COUNT TO 1 + MOVE CURRENT FIELD NUMBER
MOVE CURRENT FIELD NO. TO DIAGNOSTIC
SET SERIOUS TYPE ERROR
EXPRRE
SET SERIOUS ERROR TYPE & ADD 1 TO ERROR COUNT
EXPEX
RESTORE REGISTERS
RETURN TO CALLING PT

CHART HC. EXVAL MAINLINE (SHEET 3 OF 3)

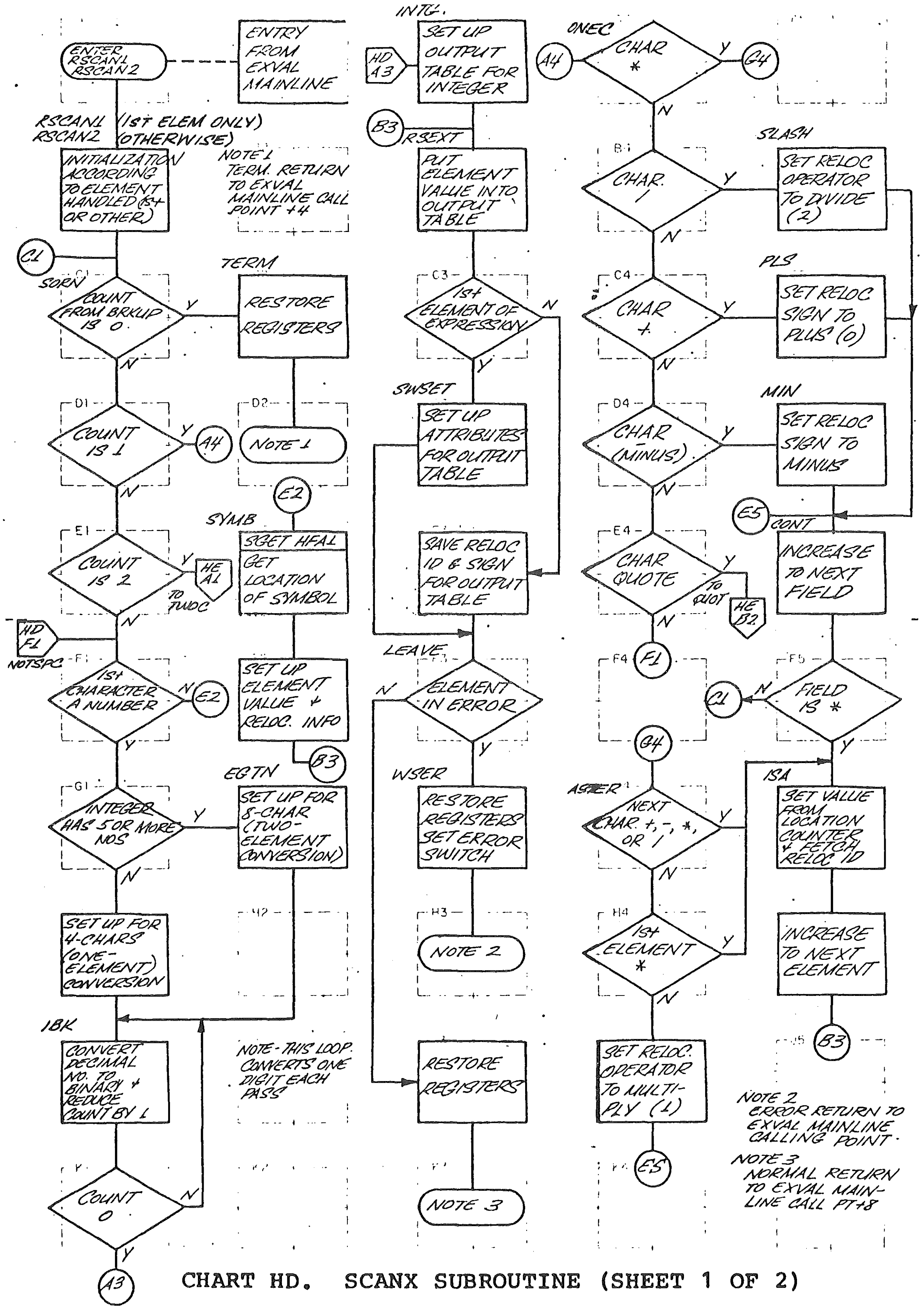

ENTER RSCAN1 RSCAN2
ENTRY FROM EXVAL MAINLINE
RSCAN1 (1ST ELEM ONLY)
RSCAN2 (OTHERWISE)
INITIALIZATION ACCORDING TO ELEMENT HANDLED (1st OR OTHER)
NOTE 1 TERM. RETURN TO EXVAL MAINLINE CALL POINT +4
C1
SORN
COUNT FROM BRKUP IS 0
TERM
RESTORE REGISTERS
COUNT IS 1
A4
NOTE 1
E2
SYMB
SGET HFAL
GET LOCATION OF SYMBOL
COUNT IS 2
HE A1
TO TWOC
HD F1
NOTSPC
1st CHARACTER A NUMBER
E2
SET UP ELEMENT VALUE + RELOC. INFO
B3
EGTN
INTEGER HAS 5 OR MORE NOS
SET UP FOR 8-CHAR (TWO-ELEMENT CONVERSION)
SET UP FOR 4-CHARS (ONE-ELEMENT) CONVERSION
IBK
CONVERT DECIMAL NO. TO BINARY + REDUCE COUNT BY 1
NOTE - THIS LOOP CONVERTS ONE DIGIT EACH PASS
COUNT 0
A3
INTG.
HD A3
SET UP OUTPUT TABLE FOR INTEGER
B3
RSEXT
PUT ELEMENT VALUE INTO OUTPUT TABLE
1st ELEMENT OF EXPRESSION
SWSET
SET UP ATTRIBUTES FOR OUTPUT TABLE
SAVE RELOC ID & SIGN FOR OUTPUT TABLE
LEAVE
ELEMENT IN ERROR
WSER
RESTORE REGISTERS SET ERROR SWITCH
NOTE 2
RESTORE REGISTERS
NOTE 3
ONEC
A4
CHAR *
G4
CHAR /
SLASH
SET RELOC OPERATOR TO DIVIDE (2)
CHAR +
PLS
SET RELOC SIGN TO PLUS (0)
CHAR (MINUS)
MIN
SET RELOC SIGN TO MINUS
CHAR QUOTE
TO QUOT
HE B2
E5
CONT
INCREASE TO NEXT FIELD
F1
C1
FIELD IS *
G4
ASTER
NEXT CHAR +, -, *, OR /
ISA
SET VALUE FROM LOCATION COUNTER + FETCH RELOC ID
1st ELEMENT *
INCREASE TO NEXT ELEMENT
SET RELOC. OPERATOR TO MULTIPLY (1)
E5
B3
NOTE 2 ERROR RETURN TO EXVAL MAINLINE CALLING POINT
NOTE 3 NORMAL RETURN TO EXVAL MAINLINE CALL PT+8
Y
N

CHART HD. SCANX SUBROUTINE (SHEET 1 OF 2)

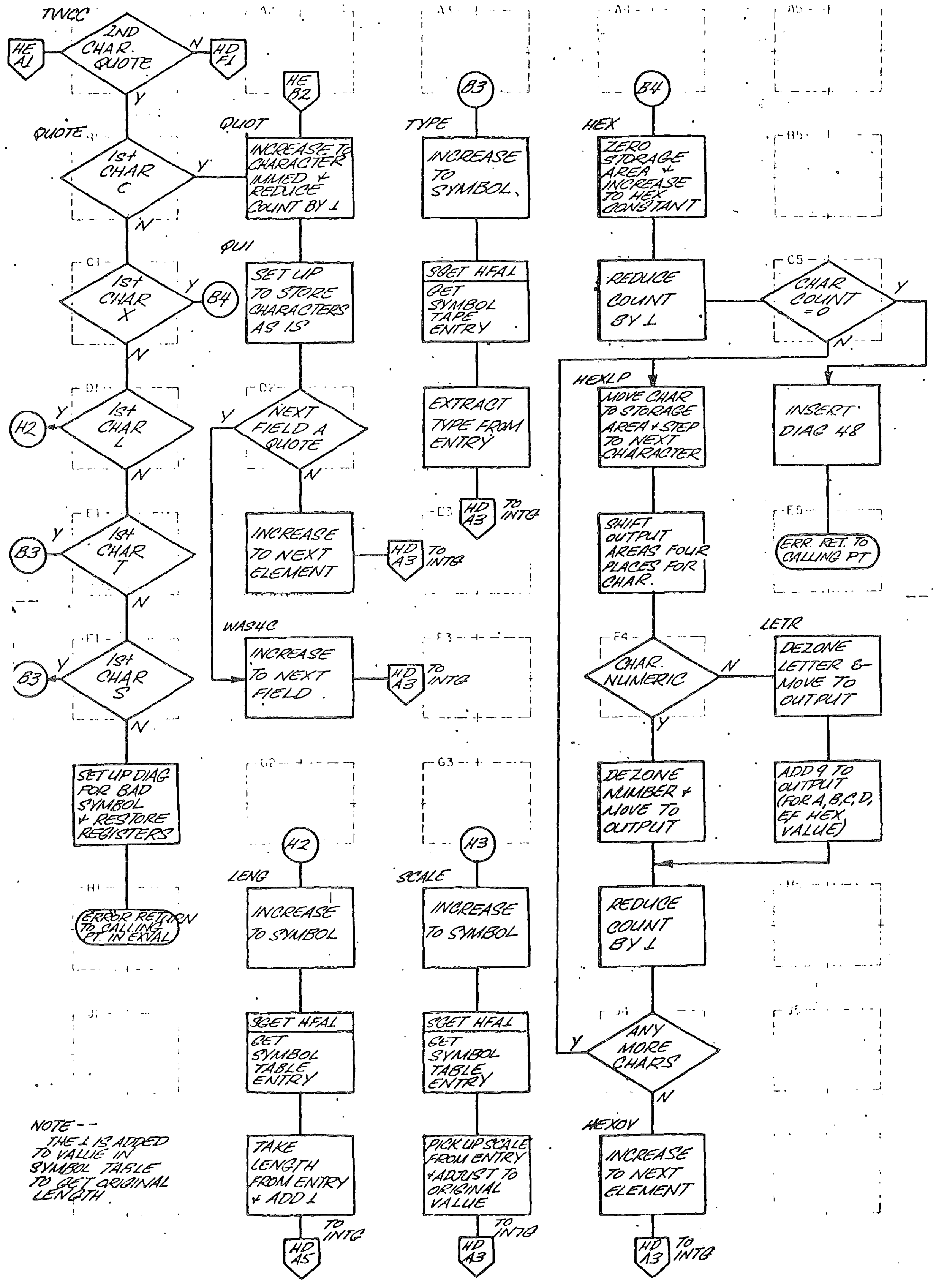


CHART HE. SCANX SUBROUTINE (SHEET 2 OF 2)

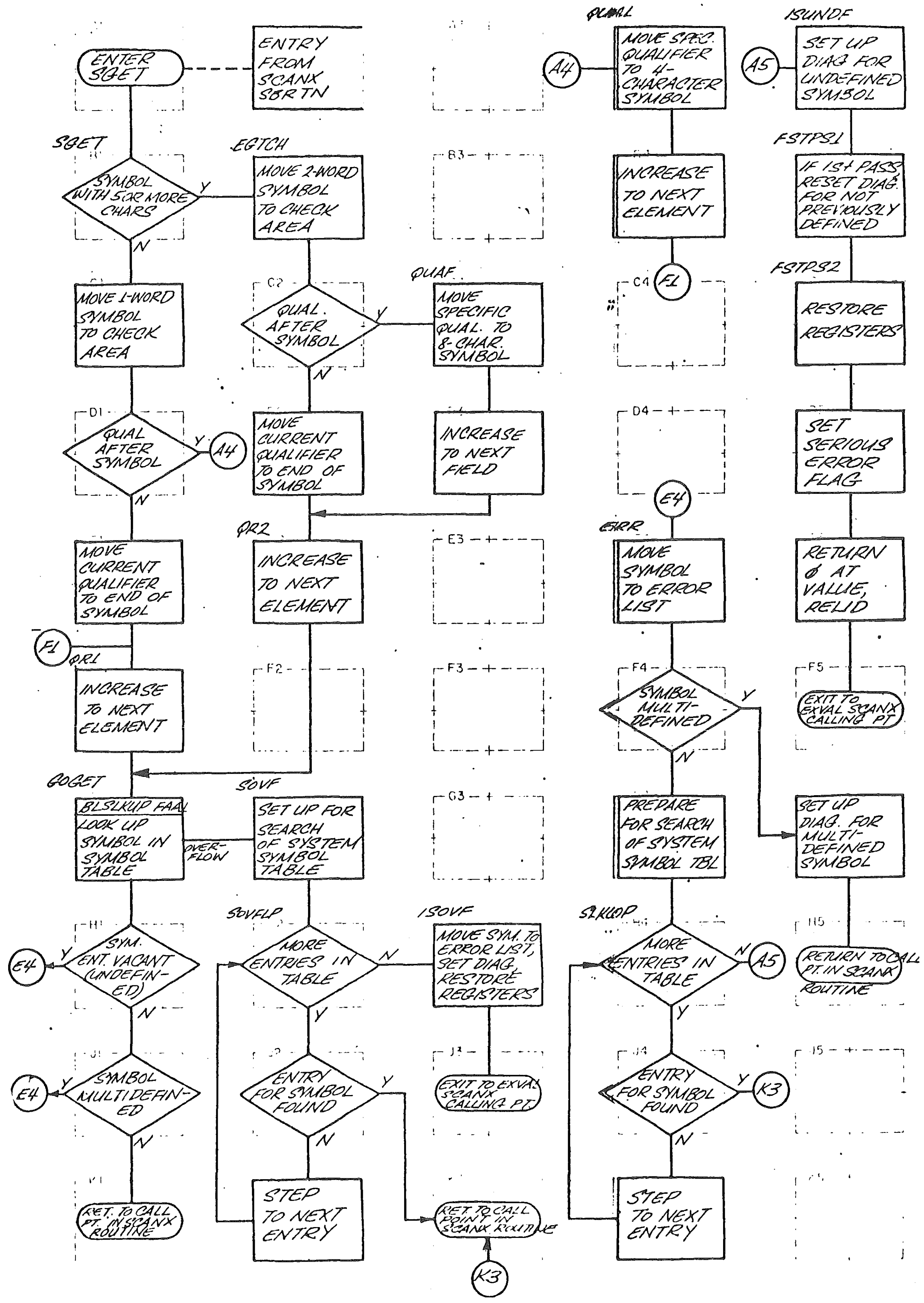
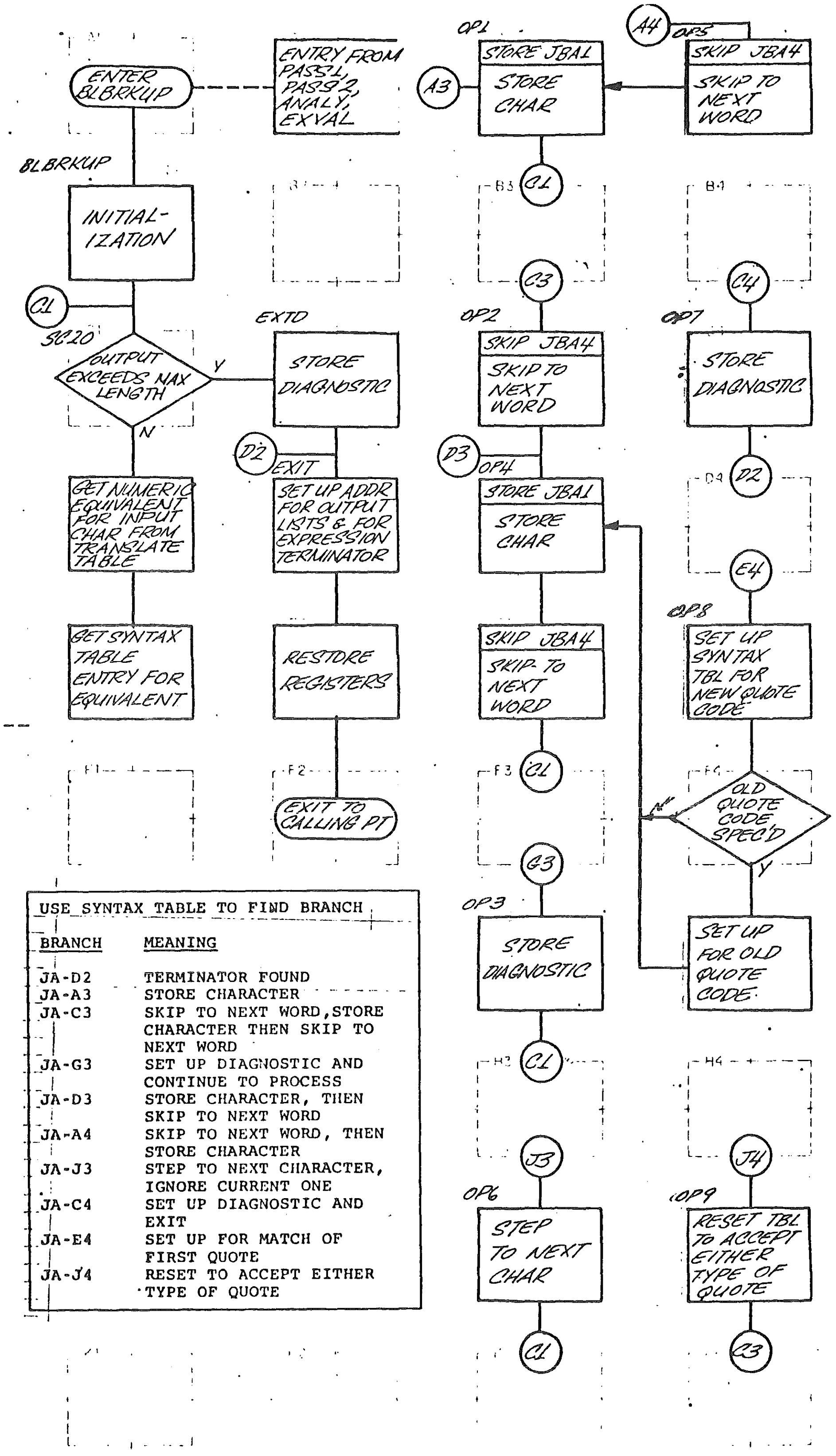

ENTER SGET
ENTRY FROM SCANX SBRTN
SGET
SYMBOL WITH 5 OR MORE CHARS
Y
N
EGTCH
MOVE 2-WORD SYMBOL TO CHECK AREA
MOVE 1-WORD SYMBOL TO CHECK AREA
QUAL. AFTER SYMBOL
QUAF
MOVE SPECIFIC QUAL. TO 8-CHAR. SYMBOL
QUAL AFTER SYMBOL
A4
MOVE CURRENT QUALIFIER TO END OF SYMBOL
INCREASE TO NEXT FIELD
QR2
MOVE CURRENT QUALIFIER TO END OF SYMBOL
INCREASE TO NEXT ELEMENT
F1
QR1
INCREASE TO NEXT ELEMENT
GOGET
SOVF
BLSLKUP FAAL
LOOK UP SYMBOL IN SYMBOL TABLE
OVER-FLOW
SET UP FOR SEARCH OF SYSTEM SYMBOL TABLE
SOVFLP
ISOVF
SYM. ENT. VACANT (UNDEFIN-ED)
E4
MORE ENTRIES IN TABLE
MOVE SYM. TO ERROR LIST, SET DIAG. RESTORE REGISTERS
SYMBOL MULTIDEFIN-ED
ENTRY FOR SYMBOL FOUND
EXIT TO EXVAL SCANX CALLING PT.
RET. TO CALL PT. IN SCANX ROUTINE
STEP TO NEXT ENTRY
RET. TO CALL POINT IN SCANX ROUTINE
K3
QUAL
A4
MOVE SPEC. QUALIFIER TO 4-CHARACTER SYMBOL
INCREASE TO NEXT ELEMENT
F1
E4
ERR
MOVE SYMBOL TO ERROR LIST
SYMBOL MULTI-DEFINED
PREPARE FOR SEARCH OF SYSTEM SYMBOL TBL
SLKUP
MORE ENTRIES IN TABLE
A5
ENTRY FOR SYMBOL FOUND
K3
STEP TO NEXT ENTRY
ISUNDF
A5
SET UP DIAG FOR UNDEFINED SYMBOL
FSTPS1
IF 1st PASS RESET DIAG. FOR NOT PREVIOUSLY DEFINED
FSTPS2
RESTORE REGISTERS
SET SERIOUS ERROR FLAG
RETURN Ø AT VALUE, RELID
EXIT TO EXVAL SCANX CALLING PT
SET UP DIAG. FOR MULTI-DEFINED SYMBOL
RETURN TO CALL PT. IN SCANX ROUTINE

CHART HF. SGET SUBROUTINE



USE SYNTAX TABLE TO FIND BRANCH

BRANCH	MEANING
JA-D2	TERMINATOR FOUND
JA-A3	STORE CHARACTER
JA-C3	SKIP TO NEXT WORD,STORE CHARACTER THEN SKIP TO NEXT WORD
JA-G3	SET UP DIAGNOSTIC AND CONTINUE TO PROCESS
JA-D3	STORE CHARACTER, THEN SKIP TO NEXT WORD
JA-A4	SKIP TO NEXT WORD, THEN STORE CHARACTER
JA-J3	STEP TO NEXT CHARACTER, IGNORE CURRENT ONE
JA-C4	SET UP DIAGNOSTIC AND EXIT
JA-E4	SET UP FOR MATCH OF FIRST QUOTE
JA-J4	RESET TO ACCEPT EITHER TYPE OF QUOTE

CHART JA. BLBRKUP ROUTINE

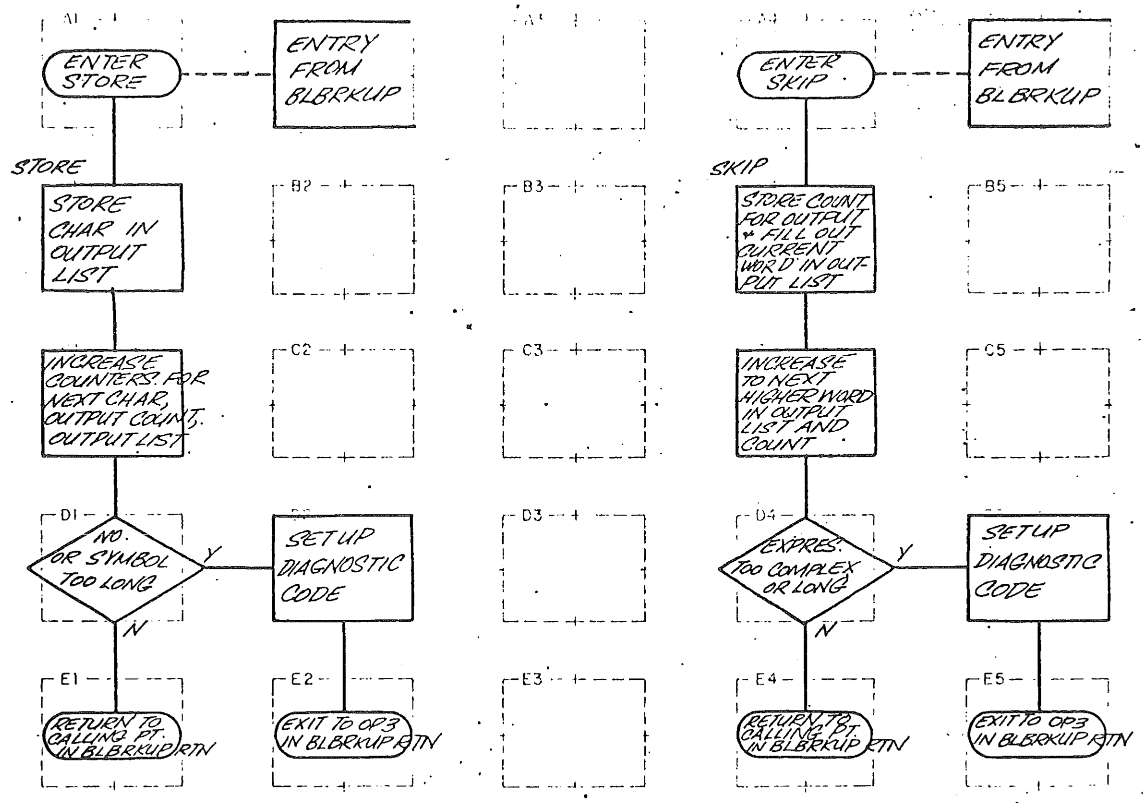

ENTER STORE
ENTRY FROM BLBRKUP
STORE
STORE CHAR IN OUTPUT LIST
INCREASE COUNTERS: FOR NEXT CHAR, OUTPUT COUNT, OUTPUT LIST
NO. OR SYMBOL TOO LONG
Y
N
SET UP DIAGNOSTIC CODE
RETURN TO CALLING PT. IN BLBRKUP RTN
EXIT TO OP3 IN BLBRKUP RTN
ENTER SKIP
ENTRY FROM BLBRKUP
SKIP
STORE COUNT FOR OUTPUT & FILL OUT CURRENT WORD IN OUTPUT LIST
INCREASE TO NEXT HIGHER WORD IN OUTPUT LIST AND COUNT
EXPRES. TOO COMPLEX OR LONG
Y
N
SET UP DIAGNOSTIC CODE
RETURN TO CALLING PT. IN BLBRKUP RTN
EXIT TO OP3 IN BLBRKUP RTN

CHART JB. STORE, SKIP SUBROUTINES

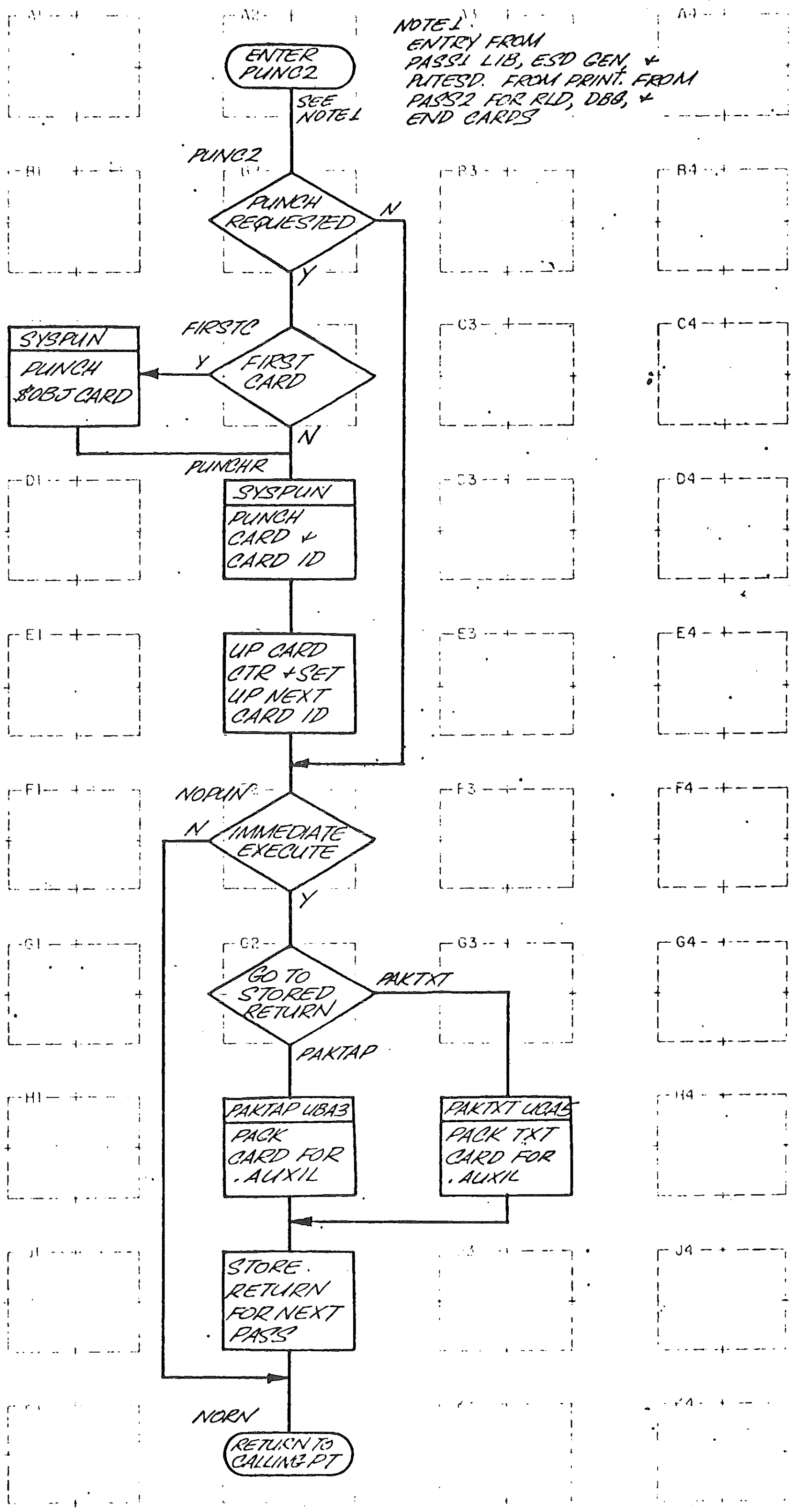


CHART UA. PUNC2 ROUTINE

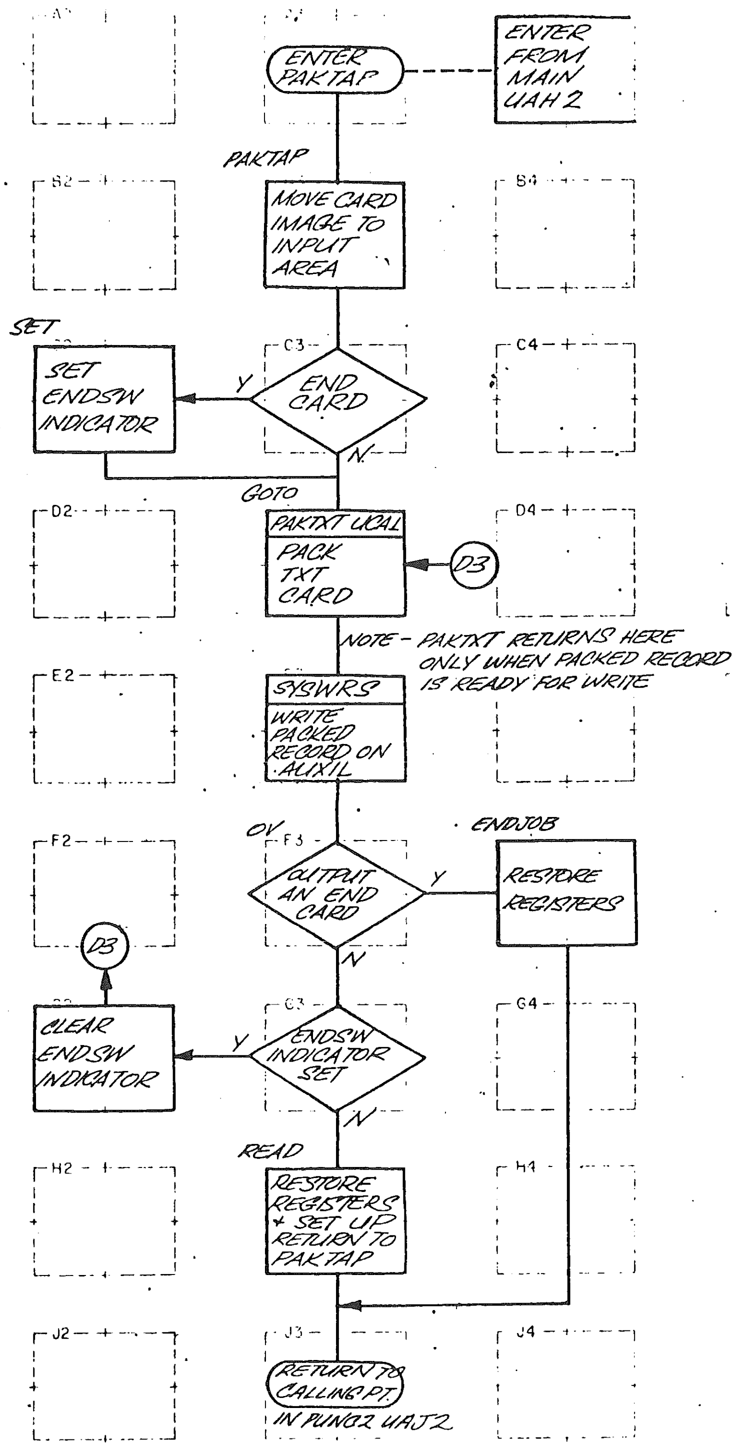

ENTER PAKTAP
ENTER FROM MAIN UAH2
PAKTAP
MOVE CARD IMAGE TO INPUT AREA
SET
SET ENDSW INDICATOR
Y
END CARD
N
GOTO
PAKTXT UCAL
PACK TXT CARD
D3
NOTE - PAKTXT RETURNS HERE ONLY WHEN PACKED RECORD IS READY FOR WRITE
SYSWRS
WRITE PACKED RECORD ON AUXIL
OV
ENDJOB
OUTPUT AN END CARD
Y
RESTORE REGISTERS
N
D3
CLEAR ENDSW INDICATOR
Y
ENDSW INDICATOR SET
N
READ
RESTORE REGISTERS + SET UP RETURN TO PAKTAP
RETURN TO CALLING PT. IN PUNO2 UAJ2

CHART UB. PAKTAP SUBROUTINE

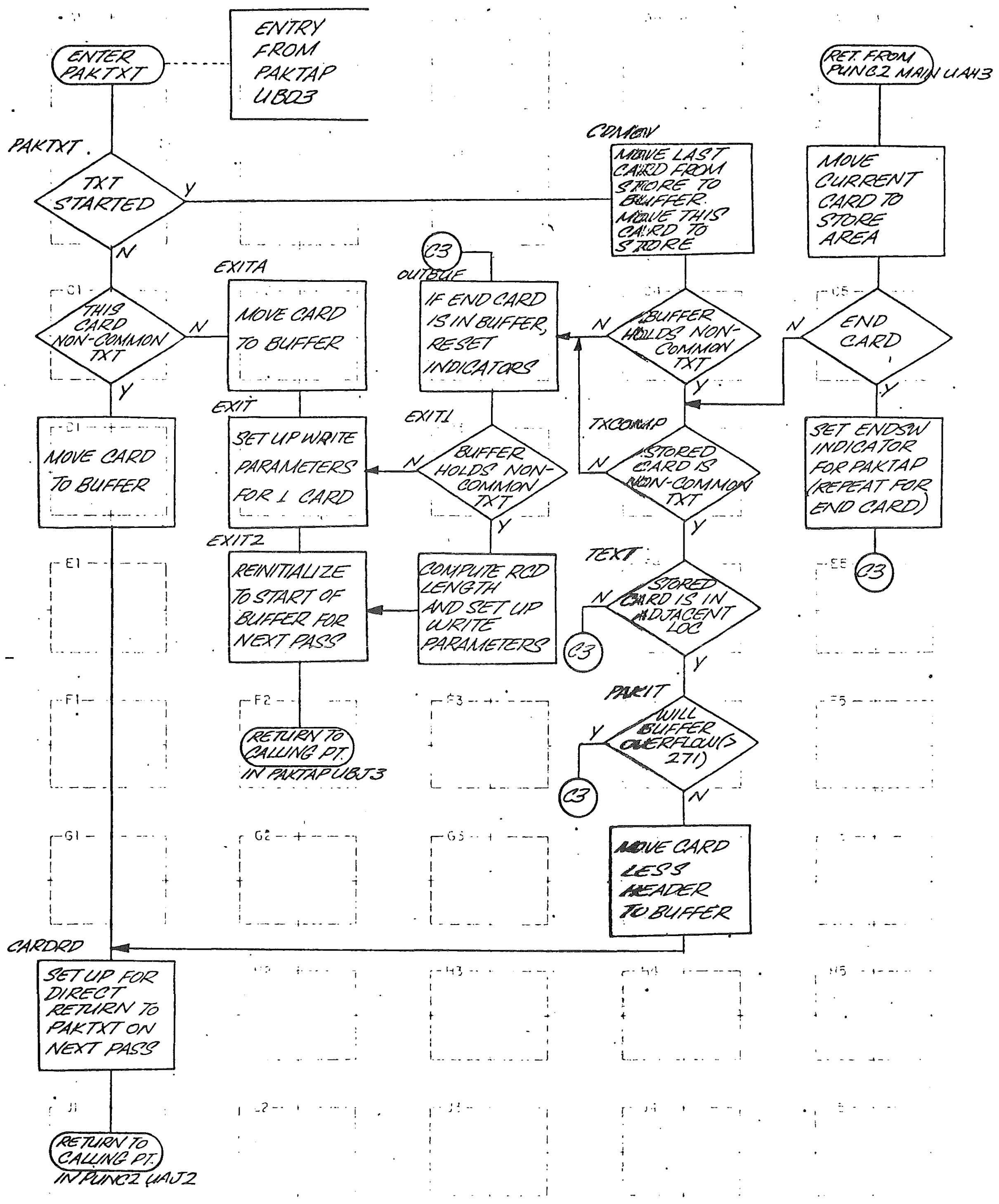

ENTER PAKTXT
ENTRY FROM PAKTAP UB03
RET FROM PUNC2 MAIN UAH3
PAKTXT
TXT STARTED
Y
N
COMMON
MOVE LAST CARD FROM STORE TO BUFFER. MOVE THIS CARD TO STORE
MOVE CURRENT CARD TO STORE AREA
C3
OUTBUF
EXITA
THIS CARD NON-COMMON TXT
MOVE CARD TO BUFFER
IF END CARD IS IN BUFFER, RESET INDICATORS
BUFFER HOLDS NON-COMMON TXT
END CARD
EXIT
EXIT1
TXCOMMP
MOVE CARD TO BUFFER
SET UP WRITE PARAMETERS FOR 1 CARD
BUFFER HOLDS NON-COMMON TXT
STORED CARD IS NON-COMMON TXT
SET ENDSW INDICATOR FOR PAKTAP (REPEAT FOR END CARD)
EXIT2
TEXT
REINITIALIZE TO START OF BUFFER FOR NEXT PASS
COMPUTE RCD LENGTH AND SET UP WRITE PARAMETERS
STORED CARD IS IN ADJACENT LOC
PAKIT
RETURN TO CALLING PT. IN PAKTAP UBJ3
WILL BUFFER OVERFLOW(> 271)
MOVE CARD LESS HEADER TO BUFFER
CARDRD
SET UP FOR DIRECT RETURN TO PAKTXT ON NEXT PASS
RETURN TO CALLING PT. IN PUNC2 UAJ2

CHART UC. PAKTXT SUBROUTINE

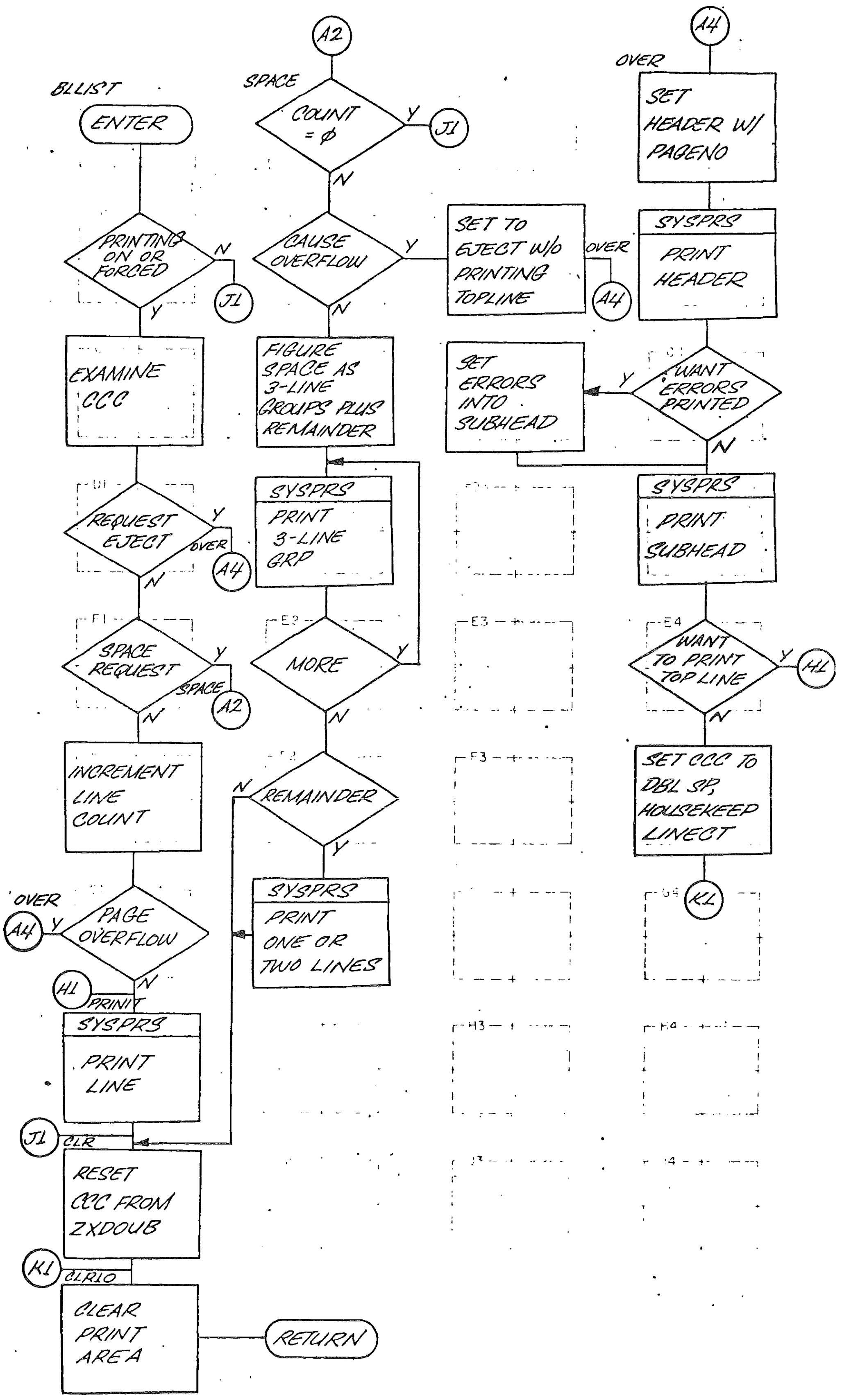


CHART VA. BLLIST ROUTINE

3.0 PASS 2 DESCRIPTION

a. Function

Pass 2 of the IBM 9020 Assembler produces object language from the statements processed and tables built up by pass 1. Pass 2 also builds the RLD table and converts storage addresses into the base-displacement form required in object language. As determined by the job control card, pass 2 may also prepare the program listing, the object deck, and object text for immediate execution.

b. Organization

Pass 2 of the assembler is executed by the BLPAS2 routine which uses some of the routines loaded with the BLPAS1 routines from BAL. Those routines not described in the preceding section of this manual are described in this section.

3.1 BLPAS2

The BLPAS2 routine, the main body of the assembler second pass, consists of a mainline section that divides into three routines: new statement, machine operation, and pseudo-operation. (See Figure 3-1.) In addition, the BLPAS2 program includes a series of independent subroutines.

After initialization by the INIT3 subroutine in BAL, BLPAS2 processes one record during each iteration. In the new statement routine, the statement is obtained from the intermediate buffer (or .WORK2 after the buffer is emptied) and checked for type. A comment or ignore statement is printed immediately (if a listing was requested); a diagnostic record is stacked for issuance after the statement to which it applies. A literal reference record is saved to obtain the attributes of the literal in the following statement.

For a BAL statement (which meets none of the above tests), look up of the operation code determines whether it calls for a machine operation or a pseudo-operation. If the statement is a machine operation, the operands are assembled by appropriate subroutines, the statement is printed and/or punched if requested, and the instruction length is added to the location counter, followed by a return to get the next statement. If the statement is a pseudo-operation, the proper pseudo-operation processing subroutine is called. Upon completion of the selected subroutine, the statement is printed and/or punched before return is made to get the next statement.

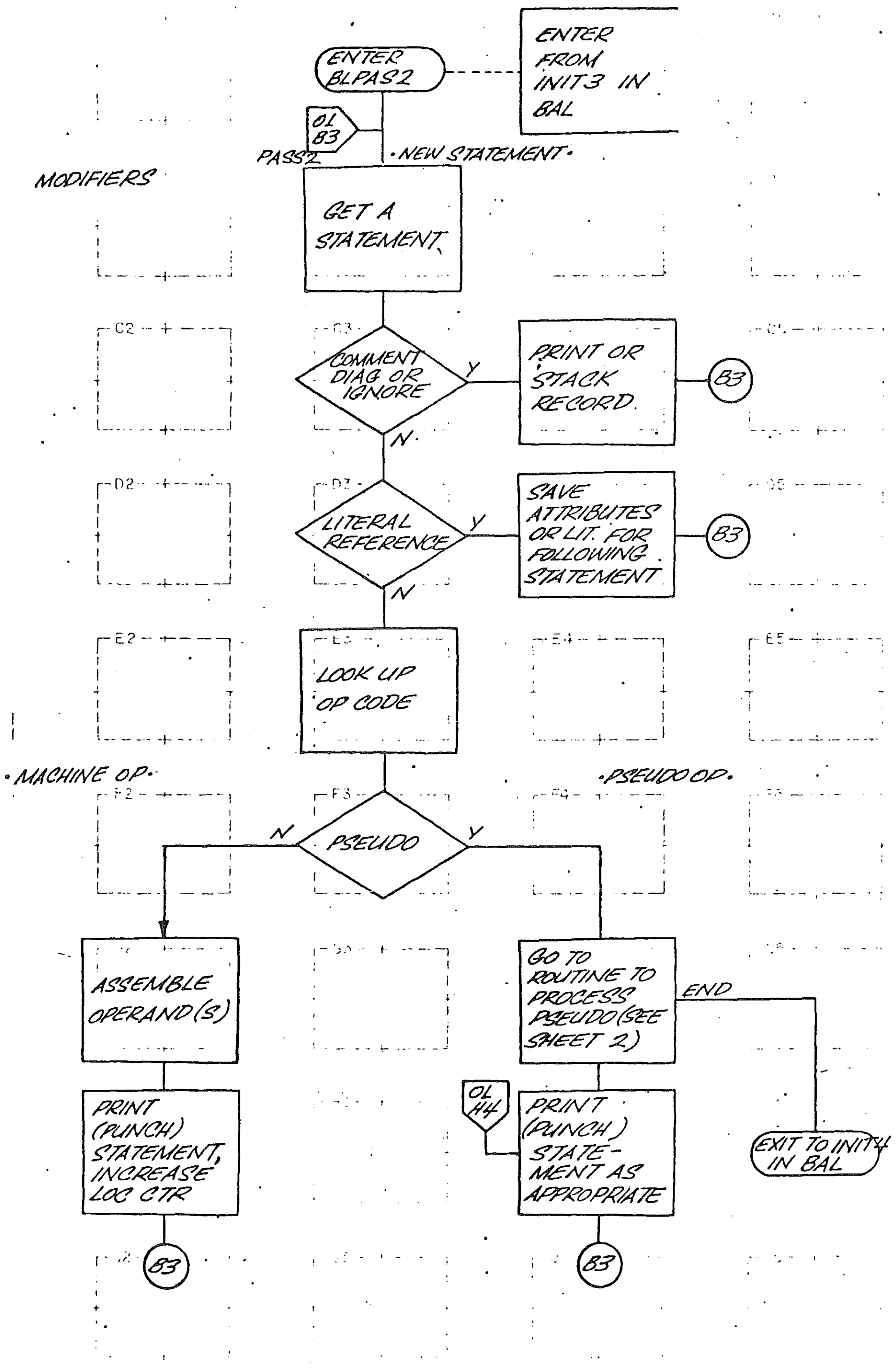


FIGURE 3-1. PASS 2 (SHEET 1 OF 2)

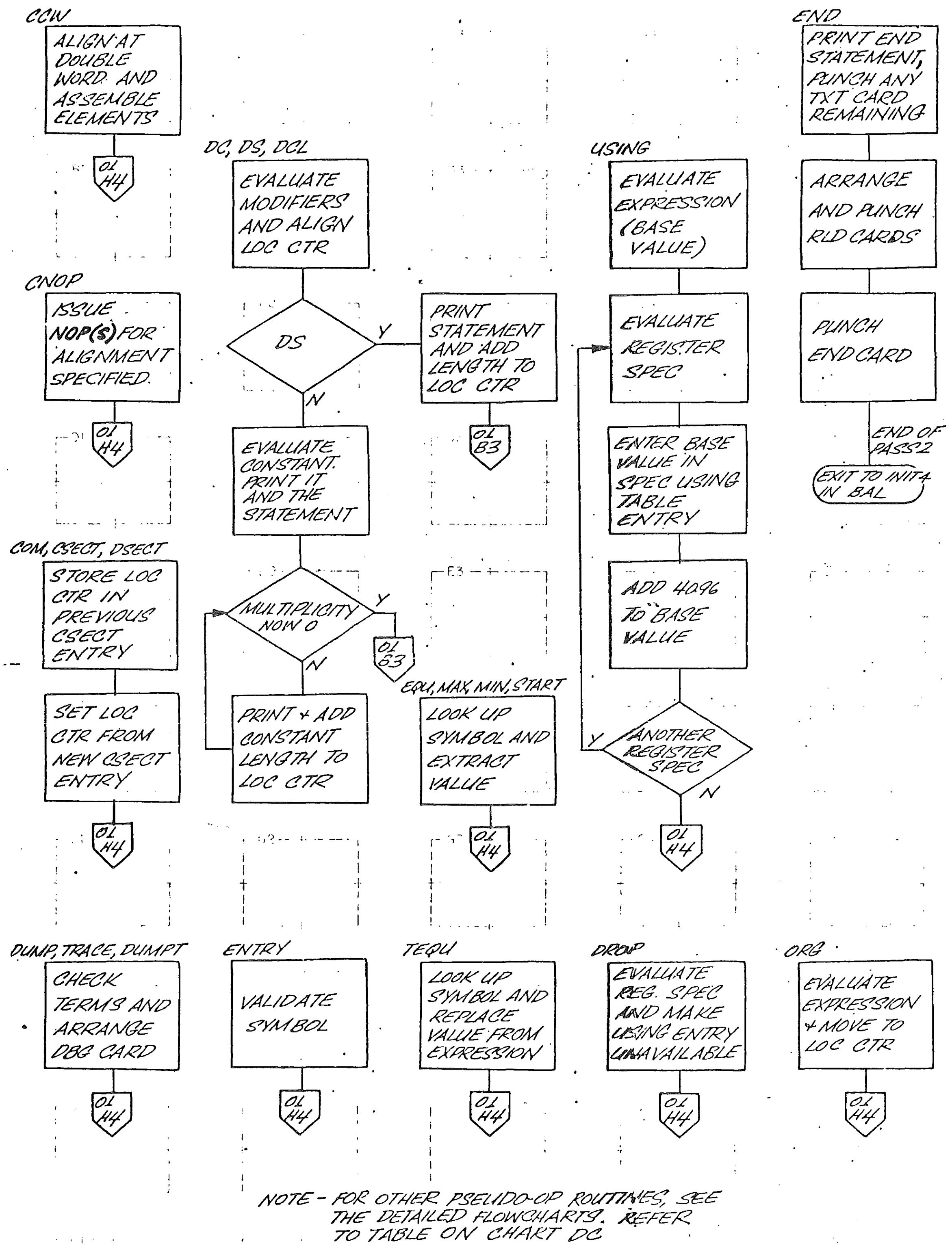

CCW
ALIGN AT DOUBLE WORD AND ASSEMBLE ELEMENTS
01 H4
CNOP
ISSUE NOP(S) FOR ALIGNMENT SPECIFIED.
01 H4
COM, CSECT, DSECT
STORE LOC CTR IN PREVIOUS CSECT ENTRY
SET LOC CTR FROM NEW CSECT ENTRY
01 H4
DC, DS, DCL
EVALUATE MODIFIERS AND ALIGN LOC CTR
DS
Y
N
PRINT STATEMENT AND ADD LENGTH TO LOC CTR
01 B3
EVALUATE CONSTANT. PRINT IT AND THE STATEMENT
MULTIPLICITY NOW 0
Y
N
01 B3
PRINT + ADD CONSTANT LENGTH TO LOC CTR
EQU, MAX, MIN, START
LOOK UP SYMBOL AND EXTRACT VALUE
01 H4
USING
EVALUATE EXPRESSION (BASE VALUE)
EVALUATE REGISTER SPEC
ENTER BASE VALUE IN SPEC USING TABLE ENTRY
ADD 4096 TO BASE VALUE
ANOTHER REGISTER SPEC
Y
N
01 H4
END
PRINT END STATEMENT, PUNCH ANY TXT CARD REMAINING
ARRANGE AND PUNCH RLD CARDS
PUNCH END CARD
END OF PASS 2
EXIT TO INIT4 IN BAL
DUMP, TRACE, DUMPT
CHECK TERMS AND ARRANGE DBG CARD
01 H4
ENTRY
VALIDATE SYMBOL
01 H4
TEQU
LOOK UP SYMBOL AND REPLACE VALUE FROM EXPRESSION
01 H4
DROP
EVALUATE REG. SPEC AND MAKE USING ENTRY UNAVAILABLE
01 H4
ORG
EVALUATE EXPRESSION + MOVE TO LOC CTR
01 H4
NOTE - FOR OTHER PSEUDO-OP ROUTINES, SEE THE DETAILED FLOWCHARTS. REFER TO TABLE ON CHART DC

FIGURE 3-1. PASS 2 (SHEET 2 OF 2)

The pseudo-operation subroutines (shown in Figure 3-1) each process one type or several related types of pseudo-operation statements. Of particular note, the END statement signals the end of the source program. When it is encountered, the END subroutine puts out the Relocation List Dictionary (RLD), then exits to INIT4 in BAL to terminate BLPAS2.

3.1.1 BLPAS2 Mainline

3.1.1.1 New Statement Routine (PASS2)

FUNCTION: This routine (Chart OA) obtains a source statement and determines whether it is a machine operation or a pseudo-operation.

ENTRY: This routine is first entered at BLPAS2 from INIT3 in BAL. Each successive iteration re-enters at PASS2.

OPERATION: This routine calls BLIOGET2 to get a statement from the intermediate buffer (or from .WORK2 after the buffer is emptied). A comment is passed to the COMMEN subroutine, a diagnostic is stacked by the DIAG routine, or an ignore record is passed to the BLPRINT routine. A literal reference record is stored for use in assembling the literal in the following source statement. Return is made after handling any of these four types to PASS2 for the next statement.

For a machine- or pseudo-operation statement, the operation code is checked for valid length, then submitted for a search of the operation code table by the BLOPLKUP routine.

EXIT: If the operation code is a pseudo-operation, this routine branches to PSEUDO in the pseudo-operation routine. If the operation code is a machine operation, this routine falls through to the machine operation routine.

ERRORS: If an overlong operation code is encountered, this routine calls the ILLOP subroutine to process the error.

3.1.1.2 Machine Operation Routine

FUNCTION: This routine (Chart OB) assembles the symbolic operands of the machine operation statement into object language along with the object operation code from the operation code table, then prints and/or punches the statement and object text.

ENTRY: This routine is entered from the new statement routine. Entry is also made at NOMOR from the EXBC and EXBCR subroutines.

OPERATION: This routine first edits the object operation code (taken from the operation code table), checks location counter alignment, and gets the operand count for this instruction from the operation code table.

Next, for each operand, the routine determines the required operand type from the operation code table and calls the appropriate operand edit subroutine. After the operands have been assembled and their terminators checked, a check is made for a privileged operation code before the statement is printed and punched. Finally, the instruction length is added to the location counter.

EXIT: This routine exits to PASS2 in the new statement routine.

ERRORS: If the location counter is not aligned to a halfword boundary, diagnostic 16 is issued and alignment is made before the routine proceeds.

An invalid terminator for the first or second operand bypasses processing of any succeeding operands and diagnostic 14 is issued.

If a privileged operation code is encountered, diagnostic 13 is issued.

If no pointer is found in the operation code table, exit is made to SYSTER terminating the assembly operation.

If the location counter overflows, the overflow is fixed (erased) and diagnostic 77 is issued for the next statement.

COMMENTS: Errors in the source statement operands are handled by the operand edit subroutines and described with those subroutines.

3.1.1.3 Pseudo-Operation Routine

FUNCTION: This routine (Chart OC) selects the proper subroutine to process a pseudo-operation statement.

ENTRY: This routine is entered at PSEUDO from the new statement routine.

OPERATION: This routine uses a pointer in the MASTER operation code table to select the proper subroutines. These subroutines are described separately below.

EXIT: This routine exits to PASS2 in the new statement routine.

ERRORS: If a disallowed pseudo-operation code is found or an operation code not in the operation code table is found, this routine calls the ILLOP subroutine.

3.1.1.4 EJECT Subroutine

FUNCTION: This subroutine causes the listing to skip to a new page, by calling the BLLIST routine to space X'3F' lines.

ENTRY: This subroutine is entered at EJECT from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.1.5 SPACE Subroutine

FUNCTION: This subroutine (Chart OC) causes the listing to space the specified number of lines (less than a full page) by calling the BLLIST routine.

ENTRY: This subroutine is entered at SPACE from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

COMMENTS: No diagnostics are issued even if the operand contains errors.

3.1.1.6 ISEQ and SSEQ Subroutines

FUNCTION: These subroutines initiate or suppress sequence number checking.

ENTRY: These subroutines are entered at ISEQ or SSEQ, respectively, from the pseudo-operation routine.

EXIT: These subroutines exit to PASS2 in the new statement routine.

3.1.1.7 NLIST and LIST Subroutines

FUNCTION: These subroutines temporarily suppress or resume listing.

ENTRY: These subroutines are entered at NLIST or LIST, respectively, from the pseudo-operation routine.

EXIT: These subroutines exit to PASS2 in the new statement routine.

3.1.1.8 SPEM and RPEM Subroutines

FUNCTION: These subroutines suppress or resume the printing of possible-error messages with selective suppression of privileged operation, void expression, and all other possible error messages (except the previously named two).

ENTRY: These subroutines are entered at SPEM or RPEM, respectively, from the pseudo-operation routine.

EXIT: These subroutines exit to PASS2 in the new statement routine.

3.1.1.9 IGNORE Subroutine

FUNCTION: This subroutine handles statements that are ignored by BLPAS2 (e.g., EXTRN, LIB), calling the BLPRINT routine for listing.

ENTRY: This subroutine is entered at IGNORE from the pseudo-operation routine.

EXIT: This subroutine exits to the new statement routine at PASS2.

3.1.1.10 MISPL Subroutine

FUNCTION: This subroutine handles statements that are disallowed by the system or that should not have reached BLPAS2 (e.g., ICTL), by issuing diagnostic 15.

ENTRY: This subroutine is entered at MISPL from the pseudo-operation routine.

EXIT: This subroutine exits to the new statement routine at PASS2.

3.1.1.11 CCW Subroutine

FUNCTION: This subroutine (Chart OD) validates and assembles a CCW.

ENTRY: This subroutine is entered at CCW from the pseudo-operation routine.

OPERATION: This subroutine begins by aligning the location counter at a doubleword boundary. It then validates and moves to the object area the CCW command code, data address, flags, and data count. The data address is also entered in the RLD table if it is relocatable. Finally, the CCW statement is submitted to the BLPRINT routine and the doubleword length is added to the location counter.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: A bad terminator following any element of the CCW bypasses processing of all succeeding elements and diagnostic 14 is issued.

An error in an element prevents its assembly into the CCW (with diagnostics issued by the validation subroutine) with two exceptions.

1. A nontruncated flag byte (last 3 bits not 0) is truncated before assembly and diagnostic 9 is issued.

2. A nonabsolute data count is assembled and diagnostic 54 is issued.

If entry of the data address in the RLD table causes overflow, diagnostic 45 is issued.

If addition of the CCW length to the location counter causes its overflow, the overflow is 'fixed' (high-order byte zeroed) by the LOGOVF, which issues diagnostic 77 for the succeeding statement, indicating an incorrect location counter for that statement.

3.1.1.12 EXBC and EXBCR Subroutines

FUNCTION: These subroutines (Chart OD) assemble extended mnemonic branch instructions.

ENTRY: These subroutines are entered at EXBC or at EXBCR from the pseudo-operation routine.

OPERATION: These subroutines begin by moving the operation code and condition (first operand) from the operation code table entry to the object area, then check the alignment of the location counter. Finally, the subroutines call the appropriate operand edit subroutine for its branch address.

EXIT: These subroutines exit to NOMOR in the machine operation routine to check the operand terminator, print the statement, and add the instruction length to the location counter.

ERRORS: If the location counter is not aligned to a halfword boundary, diagnostic 16 is issued and the HALF subroutine is called to perform the alignment.

Any errors in the operand are handled by the edit subroutine.

3.1.1.13 CNOP Subroutine

FUNCTION: This subroutine (Chart OE) assembles one or more NOP instructions, as needed, to align the location counter as specified by the CNOP subroutine.

ENTRY: This subroutine is entered at CNOP from the pseudo-operation routine.

OPERATION: This subroutine begins by aligning to a halfword boundary if required (using the HALF subroutine), then calls BLEXVAL twice to evaluate the two CNOP operands. With the desired alignment now available, the location counter is checked to determine the alignment increment required. Then, as required, a 4-byte NOP (BC format) is put out, the length is added to the location counter, and the CNOP statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If addition of the NOP instruction length to the location counter causes overflow, the overflow is 'fixed' and a diagnostic is stacked for issuance with the next statement.

3.1.1.14 COM Subroutine

FUNCTION: This subroutine (Chart OE) initiates or resumes the common section, first saving the location counter in the current CSECT entry.

ENTRY: This subroutine is entered at COM from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.1.15 CSECT and DSECT Subroutines

FUNCTION: These subroutines (Chart OF) initiate or resume the control section named in the statement.

ENTRY: These subroutines are entered at CSECT or at DSECT (depending on the statement operation code) from the pseudo-operation routine.

OPERATION: These subroutines first save the location counter in the current CSECT-EXTRN table entry, then search that table for the entry named in the statement. If no match is found, the search is repeated for an entry with a blank name. Next, the entry found is checked for whether it is CSECT or a DSECT, and the DSECT switch is set accordingly. Finally, the location counter is set from the table entry and the statement is printed.

EXIT: These subroutines exit to PASS 2 in the new statement routine.

COMMENTS: Since the BLPAS1 program allows a DSECT statement to resume a CSECT entry and vice versa (issuing a diagnostic to call attention to the occurrence), these subroutines must allow for the same possibility; i.e., that a DSECT statement may be resuming a CSECT or vice versa.

3.1.1.16 DCODS Subroutine

FUNCTION: This subroutine (Charts OG, OH) handles a DC, DS, or DCL statement to reserve storage space for its operand, and, if appropriate, set up object code for the constant value.

ENTRY: This subroutine is entered at DCODS from the pseudo-operation routine.

OPERATION: This subroutine begins by calling BLCONMOD to evaluate the constant modifiers. The alignment code from BLCONMOD is then used to align the location counter for the constant. A DS statement is simply printed (and given to BLPUNC2 which will punch the current card) and the length of the constant is added to the location counter without assembling object code for loading into storage.

For a DC or DCL statement (the latter issued by BLPAS1 to define a literal), the multiplicity is checked for nonzero. Next, BLCONVAL is called to evaluate the constant. Its value is then assembled as object code, and punched, and the constant length is added to the location counter. If the multiplicity was greater than 1, the object code is repeated and its length added to the location counter as many times as called for by the multiplicity. If it is an adcon, each iteration is re-evaluated after stepping the location counter.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If a DC or DCL statement lacks a value list, diagnostic 19 is issued and the statement is handled as if it were a DS statement (no object code but storage space allocated).

If a DC statement with zero multiplicity is encountered, the statement is printed without assembly to object code or reservation of storage.

If BLCONVAL finds errors in the constant, its diagnostic messages are issued but processing continues normally.

If the location counter overflows at any point, the overflow is 'fixed' and a diagnostic is stacked for issuance with the next statement or line on the listing.

3.1.1.17 DROP Subroutine

FUNCTION: This subroutine (Chart OI) validates the register operand(s) of a DROP statement, then removes the designated register(s) from the USING table and base-displacement address calculations.

ENTRY: This subroutine is entered at DROP from the pseudo-operation routine.

OPERATION: This subroutine begins by calling the ABS8 subroutine to validate the register specification, then checks that the specified register was marked in the USING table as in use for base-displacement calculations. Finally, the table entry is marked as unavailable and, if no further registers are specified in the DROP statement, the statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the register specification is in error, diagnostic 21 is issued and no check is made of the USING table.

If the specified register was not in use, for base-displacement calculations, diagnostic 20 is issued.

3.1.1.18 DUMPT Subroutine

FUNCTION: This subroutine (Charts OS, OT) validates a tape dump request and assembles the DBG card for it.

ENTRY: This subroutine is entered at SUMPT from the pseudo-operation routine.

OPERATION: This subroutine validates and assembles the dump format label, logical tape drive number, 'from file,' 'from record,' 'to file,' 'to record,' and 'bytes per record.' The dump format and label are extracted and validated in succession, using the BLBRKUP routine for the extraction. The next five (or six) terms in the request are evaluated in succession by the INROUT subroutine and further tested for maximum value in this subroutine before being assembled onto the DBG card. Finally, the request is identified as logical or physical, the DBG card is punched, and the tape dump statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: An error in any element of the dump request causes issuance of an appropriate diagnostic and a 'no DBG card' diagnostic (22); the dump request statement is printed but no DBG card is produced.

The diagnostics issued are as follows.

a. Format error - BLBRKUP diagnostic

b. Invalid format - number 23

c. Invalid label - number 24

d. Integer error - number 31

e. Integer too large - number 25

f. Bad terminator - number 14

A blank format specification is forced as hexadecimal (HEX) format.

3.1.1.19 DUMP Subroutine

FUNCTION: This subroutine (Charts OJ, OK, OL) validates a storage and/or register dump request and assembles the DBG card for it.

ENTRY: This subroutine is entered at DUMP from the pseudo-operation routine.

OPERATION: This subroutine validates and assembles the dump format, label, 'from loc,' and 'to loc,' then determines the dump request type to select further processing.

The dump format and label are extracted and validated in succession, using the BLBRKUP routine for the extraction. If a 'from loc' is specified (indicating storage dump as well as register dump), the EXVR subroutine is called to evaluate it. If the 'from loc' is absolute, a special subroutine is entered to ensure that the 'to loc' is also absolute. Similarly, a check is made that both the 'from loc' and 'to loc' are or are not in common. In either case, EXVR is used to evaluate the 'to loc.'

After the storage dump addresses (if present) have been assembled, the dump type is examined to determine further processing if any. An emergency dump request (DUMPE) is finished at this point. An unconditional dump request (DUMP) is completed by evaluating the 'point of dump' address. The two types of conditional dump request (DUMPC and DUMPR) are each handled by a separate subroutine that processes the conditions specified in the request.

There are two subroutines within the DUMP subroutine: CONVT and GTLOC. The CONVT subroutine converts a 3-byte address into EBCDIC characters. The GTLOC subroutine evaluates the symbolic 'point of dump'.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: An error in any element of the dump request causes issuance of an appropriate diagnostic and a 'no DBG card' diagnostic (22); the dump request statement is printed but no DBG card is produced.

The diagnostics issued are as follows.

a. Format error - BLBRKUP diagnostic

b. Invalid format - number 23

c. Invalid label - number 24

d. Bad address - number 26

e. Bad integer - number 25

f. Bad condition - number 27

g. Bad register - number 28

A blank format specification is forced as HEX format.

3.1.1.20 END Subroutine

FUNCTION: This subroutine (Charts OM, ON) validates the 'begin' location if specified in the END statement, puts out all entries in the RLD table, and exits to INIT4, ending BLPAS2.

ENTRY: This subroutine is entered at END from the pseudo-operation routine.

OPERATION: This subroutine begins by calling the print subroutine to punch the final TXT card. It then calls the EXVR subroutine to evaluate the 'begin' location. If the location was defined in a control section, its value is moved to the END card. If not and the location is not absolute, BLBRKUP is called to extract the symbol (an EXTRN) for moving to the END card. The END statement is then printed.

If any entries are present in the RLD table, the table entries are printed and RLD cards are punched. Finally, the last RLD card is punched whether full or not and the END card is then punched. The sequence number of the END card is saved for the XREF routine.

The PRRLD subroutine, included in the END subroutine, prints a line of the RLD table each time it is called, if a listing has been requested.

EXIT: This subroutine exits to INIT4 in BAL, ending BLAPS2.

ERRORS: Any errors found in the 'begin' location by EXVR cause BLEXVAL diagnostics to be issued, followed by diagnostic 30, and assembly of the 'begin' location to be omitted.

If the 'begin' location is absolute, diagnostic 30 is issued and the location is not assembled on the END card.

3.1.1.21 ENTRY Subroutine

FUNCTION: This subroutine (Chart OI) evaluates the ENTRY symbol(s) and determines that the symbol has been defined in a control section or in common.

ENTRY: This subroutine is entered at ENTRY from the pseudo-operation routine.

OPERATION: This subroutine calls the EXVR subroutine to evaluate the symbol. If the expression contains no errors and identifies a location defined in a control section or in common, the ENTRY symbol is accepted. After all symbols in the ENTRY operand field have been validated, the statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the entry symbol was defined in a dummy control section, diagnostic 62 is issued.

If the last symbol is not terminated by a blank character, diagnostic 14 is issued.

COMMENTS: Errors found by BLEXVAL (called by the EXVR subroutine) cause diagnostics to be stacked before return is made to this subroutine.

3.1.1.22 PRNT Subroutine

FUNCTION: This subroutine (Chart OO) validates the operands of a PRINT statement to determine how much of each assembled constant (from a DC statement) will be listed and whether literals will be listed.

ENTRY: This subroutine is entered at PRNT from the pseudo-operation routine.

OPERATION: This subroutine begins by calling the BLBRKUP routine to extract and validate the first operand. If the operand is void or 'DATA,' constants will be printed completely in the listing; if the operand is 'NODATA,' only the first 16 bytes of each constant will be printed.

This subroutine then calls BLBRKUP again to extract and validate the second operand. If the operand is void or 'LIT,' all literals will be printed; if the operand is 'NOLIT,' literals will not be printed when the literal table is put out. The PRINT statement is then printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If errors are found by BLBRKUP in either operand, the BLBRKUP diagnostic(s) is issued, followed by diagnostic 31; further operand processing is bypassed.

A bad terminator of the first operand allows the operand to be processed, but diagnostic 31 is issued and the second operand is ignored.

3.1.1.23 TITLE Subroutine

FUNCTION: This subroutine (Chart OO) validates the punch ID (if present on the TITLE card) and causes the TITLE page of the listing to be printed.

ENTRY: This subroutine is entered at TITLE from the pseudo-operation routine.

OPERATION: If a punch ID is present on the TITLE card, this subroutine calls BLBRKUP to extract it. If valid, the ID is moved to the card sequence number field.

The subroutine then picks up the title, prints the statement, and causes the listing to skip to a new page.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the punch ID is invalid, diagnostic 32 is issued and the ID is not moved to the card sequence number field.

3.1.1.24 TRACE Subroutine

FUNCTION: This subroutine (Charts OP, OQ) validates a trace request and assembles the DBG card for it.

ENTRY: This subroutine is entered at TRACE from the pseudo-operation routine.

OPERATION: This subroutine validates and assembles the trace label, start ('from loc'), and end ('to loc') onto a DBG card. The label is extracted from the statement by BLBRKUP and, if it is not void, is moved to the DBG card. Next, if a 'from loc' is specified, the EXVR subroutine is used to evaluate the 'from loc' and the 'to loc' in succession. If the 'from loc' is absolute, a special subroutine is entered to ensure that the 'from loc' and 'to loc' are or are not in the common section. Finally, the trace type (TRACE or TRACB) is moved to the DBG card, the card is punched, and the statement is printed.

The CVT subroutine, included in the TRACE subroutine, converts a 3-byte address into EBCDIC characters.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: An error in the label or either address in the trace request causes issuance of the appropriate diagnostic (24 for label or 26 for address), followed by a 'no DBG card' diagnostic (22); the trace request statement is printed but no DBG card is produced.

3.1.1.25 USING Subroutine

FUNCTION: This subroutine (Chart OR) validates the value and registers specified in a USING statement for base-displacement calculations.

ENTRY: This subroutine is entered at USING from the pseudo-operation routine.

OPERATION: This subroutine begins by calling the EXVR subroutine to evaluate the values specified for the first base register. If the value is valid and the expression properly terminated, the subroutine then validates the general register specified as base register. If other than register 0 is specified, the register entry in the USING table is marked as available for base-displacement calculations, the relocation ID is moved into the entry, and the value is moved into the entry. If another register is specified, the value is increased by 4096 before the register specification is validated and entry made. Finally, the USING statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the EXVR subroutine finds a value error, it puts out diagnostics selected by the BLEXVAL routine. Diagnostic 33 is then issued and no entry is made in the USING table.

If a register specification is in error, diagnostic 34 is issued and no entry is made for that specification.

If register 0 is specified with a zero value, the entry is made but diagnostic 52 is issued as well.

If register 0 is specified with a non-zero value, entry is made in register 0 for a zero value but diagnostic 53 is issued.

3.1.1.26 LTORG Subroutine

FUNCTION: This subroutine aligns the location counter to a doubleword boundary in preparation for the literals (put out as DCLs) to follow and prints the LTORG statement.

ENTRY: This subroutine is entered at LTORG from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.1.27 ORG Subroutine

FUNCTION: This subroutine evaluates the ORG statement operand, sets the location counter to that value, and prints the ORG statement.

ENTRY: This subroutine is entered at ORG from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the TEQU has been used, the ORG could be for an incorrect CSECT. In this case, diagnostic 66 will be issued and the ORG ignored.

3.1.1.28 ILLOP Subroutine

FUNCTION: This subroutine is called for an invalid operation code; it issues diagnostic 35, sets up a 4-byte NOP, and adds this length to the location counter.

ENTRY: This subroutine is entered at ILLOP from the pseudo-operation routine or from the new statement routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.1.29 QUAL Subroutine

FUNCTION: This subroutine picks up the new qualifier given in the QUAL statement and prints the statement.

ENTRY: This subroutine is entered at QUAL from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

COMMENTS: A QUAL statement containing errors would be found in BLPAS1 and marked to be ignored by BLPAS2.

3.1.1.30 TEQU Subroutine

FUNCTION: This subroutine (Chart OU) changes the location attribute of the symbol named in the TEQU statement to the value of the statement operand and prints the TEQU statement.

ENTRY: This subroutine is entered at TEQU from the pseudo-operation routine.

OPERATION: This subroutine first calls BLSLKUP to find the symbol given in the statement name field in the symbol table. Next, the EXVR subroutine is called to evaluate the operand. The value is converted for listing, both it and its symbolic attribute are moved into the location field of the symbol table entry, and the TEQU statement is printed.

EXIT: This subroutine exits to PASS2 in the new statement routine.

ERRORS: If the EXVR subroutine finds errors in the TEQU statement operand, diagnostic 55 is issued in addition to the diagnostics issued by the subroutine and the new location is not entered in the symbol table.

COMMENTS: Errors in the operand would be found by PASS1 and the TEQU statement would be marked to be ignored by PASS2.

3.1.1.31 EQMXMN Subroutine

FUNCTION: This subroutine (Chart OU) assembles the location specified by the statement START, EQU, MAX, or MIN, then prints the statement. Re-evaluation may be required if TEQU has been used.

ENTRY: This subroutine is entered at EQMXMN from the pseudo-operation routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.2 BLPAS2 Common Subroutines

Included within the BLPAS2 program are 20 subroutines that are called to perform various operations for the main body of the program. These subroutines, listed below, are each described separately.

1. COMMEN - Print a comment record

2. INROUT - Evaluate an integer

3. ABS8 - Evaluate an absolute expression

4. EXVR - Evaluate an expression

5. LOCOVF - Process location counter overflow

6. DOUB - Align location counter to doubleword boundary

7. HALF - Align location counter to halfword boundary

8. COVX - Convert fullword to EBCDIC characters.

9. USBASE - Compute base and displacement

10. REGST - Evaluate register operand

11. LITYP - Evaluate register operand (R2 = 0)

12. INT8 - Evaluate 8-bit immediate operand

13. SHFT - Evaluate shift operand

14. SA - Evaluate storage-address operand

15. XA - Evaluate indexable-storage-address operand

16. SLA4Q - Evaluate storage-address operand with 4-bit length specification

17. SLA - Evaluate storage-address operand with 8-bit length specification

18. SI8Z - Evaluate storage-address operand with 8-bit length = 0

19. SA4Z - Evaluate storage-address operand with 4-bit length = 0

20. DIAG - Stack a diagnostic

3.1.2.1 Print A Comment Record Subroutine (COMMEN)

FUNCTION: This subroutine (Chart OV) accepts comments, JOVIAL statements, and JOVIAL diagnostics, printing them if listing is requested and adding to the serious and possible error counts.

ENTRY: This subroutine is entered at COMMEN from the new statement routine.

EXIT: This subroutine exits to PASS2 in the new statement routine.

3.1.2.2 Evaluate An Integer Subroutine (INROUT)

FUNCTION: This subroutine (Chart OV) evaluates an integer and validates it.

ENTRY: This subroutine is entered at INROUT from the DUMP or DUMPT subroutine.

EXIT: This subroutine returns to the calling point if an error was found in the integer or to the instruction following the calling point (normal return) if no error was found.

3.1.2.3 Evaluate Absolute Expression Subroutine (ABS8)

FUNCTION: This subroutine (Chart OW) evaluates an absolute expression and validates it against a maximum value stipulated by the calling routine.

ENTRY: This subroutine is entered at ABS8 by the DROP and USING subroutines and by the operand subroutines REGST, SA, XA, SLA4Q, SLA, and LITYP.

OPERATION: This subroutine calls EXVR to evaluate the expression, then tests that it is absolute and less than the maximum specified by the calling routine.

EXIT: This subroutine exits to the instruction following the calling point (normal return) if no error is found; to the calling point (error return) if an error is found.

3.1.2.4 Evaluate Expression Routine (EXVR)

FUNCTION: This subroutine (Chart OW) evaluates an expression submitted to it.

ENTRY: This subroutine is entered at EXVR by the CCW, DUMP, END, ENTRY, TRACE, USING, and TEQU subroutines or by the ABS8, SA, XA, SLA4Q, and SLA subroutines.

OPERATION: This subroutine calls BLEXVAL if the expression is not a literal or gets the literal characteristics from the symbol table. This subroutine issues diagnostics for any errors found.

EXIT: This subroutine exits to the instruction following the calling point (normal return) if no error is found; to the calling point (error return) if an error is found.

3.1.2.5 Process Location Counter Overflow Subroutine (LOCOVF)

FUNCTION: This subroutine stacks diagnostic 77, calling attention to the location counter overflow, and 'fixes' the overflow by zeroing the location counter high-order byte.

ENTRY: This subroutine is entered at LOCOVF whenever a location counter overflow is encountered in BLPAS2.

EXIT: This subroutine exits to the calling point.

COMMENTS: This subroutine is usually called after listing of the entry that caused the location counter overflow (which overflow will affect the location assigned to the subsequent entry). The 'overflow' diagnostic is thus listed following the next entry, whose location is incorrect because the counter has been 'fixed.'

3.1.2.6 Align to Doubleword Subroutine (DOUB)

FUNCTION: This subroutine (Chart OX) checks, and, if necessary, aligns the location counter to a doubleword boundary.

ENTRY: This subroutine is entered at DOUB from the CCW, DCODS, or LTORG subroutine.

EXIT: This subroutine exits to the calling point.

3.1.2.7 HALF Subroutine

FUNCTION: This subroutine (Chart OX) aligns the location counter to a halfword boundary.

ENTRY: This subroutine is entered at HALF from the machine operation routine or from the EXBC, EXBCR, CNOP, or DCODS subroutine.

EXIT: This subroutine exits to the calling point.

3.1.2.8 Convert Fullword to EBCDIC Subroutine (COVX)

FUNCTION: This subroutine (Chart OX) produces an EBCDIC representation of a fullword presented to it, converting each hexadecimal digit (4 bits) into one of the characters 0 through 9 or A through F representing its value.

ENTRY: This subroutine is entered at COVX from the END, USING, TEQU, or EQMXMN subroutines or from the REGST, SA, USBASE, XA, SLA4Q, SLA, INT8, or LITYP subroutines.

EXIT: This subroutine exits to the calling point.

3.1.2.9 Compute Base and Displacement Subroutine (USBASE)

FUNCTION: This subroutine (Chart OY) searches the USING table to develop a base register specification and the smallest displacement for the effective address submitted to the subroutine.

ENTRY: This subroutine is entered at USBASE from the SA, XA, SLA4Q, or SLA subroutines.

OPERATION: This subroutine begins by calling the COVX subroutine to convert the effective address into EBCDIC characters for object listing, then searches the USING table for a base register with acceptable base value (same relocation ID and base value $\leqslant$ effective address $<$ base value $+$ 4096). The displacement is computed and saved with the register specification. If another acceptable base register is found, its displacement is compared with the previously computed one to select the smaller (or later equal) displacement. After the entire USING table has been searched, the COVX subroutine is again called to convert the register specification and displacement for object area listing.

EXIT: This subroutine exits to the second instruction following the calling point (normal return) if an acceptable base register and displacement have been found.

ERRORS: If no acceptable base register is found, diagnostic 38 is issued and return is made to the instruction following the calling point (error return).

3.1.2.10 Evaluate Register Operand Subroutine (REGST)

FUNCTION: This subroutine (Chart OZ) evaluates the register specified as an operand and determines that the specified register type is that required for the statement operation code.

ENTRY: This subroutine is entered at REGST from the machine operation routine.

EXIT: This subroutine exits to the calling point.

ERRORS: If the register specification is in error, it is not moved to the object area; diagnostics calling attention to the errors are issued by the ABS8 subroutine or by the EXVR subroutine called by it.

If the specified register type does not match that required by the operation code, the register specification is assembled into the object area but a diagnostic is issued, either 36 or 37.

If the specified register value is too large, diagnostic 54 is issued.

3.1.2.11 Evaluate Register Operand, No R2 Subroutine (LITYP)

FUNCTION: This subroutine (Chart OZ) evaluates a register operand (R1 with R2 = 0).

ENTRY: This subroutine is entered at LITYP from the machine operation routine.

EXIT: This subroutine exits to the calling point.

ERRORS: If the register specification is in error, it is not moved to the object area; diagnostics calling attention to the errors are issued by the ABS8 subroutine or by the EXVR subroutine called by it.

3.1.2.12 Evaluate 8-Bit Immediate Operand Subroutine (INT8)

FUNCTION: This subroutine (Chart OZ) evaluates an integer specifying an 8-bit immediate operand.

ENTRY: This subroutine is entered at INT8 from the machine operation routine.

EXIT: This subroutine exits to the calling point.

ERRORS: If the integer specification is in error, it is not moved to the object area; diagnostics calling attention to the errors are issued by the ABS8 subroutine or by the EXVR subroutine called by it.

3.1.2.13 Evaluate Shift Operand Subroutine (SHFT)

FUNCTION: This subroutine (Chart OZ) evaluates a shift operand.

ENTRY: This subroutine is entered at SHFT from the machine operation routine.

EXIT: This subroutine exits to the calling point unless an error is found in the XA subroutine.

ERRORS: If the base register specification developed by the XA subroutine in its determination of base displacement is not 0, diagnostic 40 is issued and the R2 specification developed there is not substituted for the base.

Errors found in the XA subroutine or in subroutines that it calls cause diagnostics to be issued for them. Certain errors cause the XA subroutine to exit to PRIVBK in the new statement routine rather than returning to this subroutine.

3.1.2.14 Evaluate Storage Address Operand Subroutine (SA)

FUNCTION: This subroutine (Chart PA) evaluates a storage address operand with an implied or specified base register.

ENTRY: This subroutine is entered at SA from the machine operation routine.

OPERATION: This subroutine first calls the EXVR subroutine to evaluate the operand to obtain the effective address. Next, the USBASE subroutine is called to compute the displacement for an implied base register. If a base register is specified in the operand, the ABS8 subroutine is used to evaluate it. Finally, the address alignment is checked.

EXIT: This subroutine exits to the calling point, except when a terminator error is encountered.

ERRORS: If the operand contains errors, the EXVR subroutine issues diagnostics and the effective address is rejected.

If the effective address cannot be converted into a displacement and base, the USBASE subroutine issues a diagnostic and the displacement and base are not assembled.

If the implied base register found by USBASE is not register 0 and a base register is specified, diagnostic 40 is issued and the specification is not used (implied base is used).

If the effective address is misaligned, diagnostic 39 is issued.

If the register specification is not properly terminated, diagnostic 14 is issued and the subroutine exits to PRIVBK in the new statement routine.

3.1.2.15 Evaluate Indexable Storage Address Subroutine (XA)

FUNCTION: This subroutine (Chart PB) evaluates an indexable storage address operand with implied or specified base register.

ENTRY: This subroutine is entered at XA from the machine operation routine or from the SHFT subroutine.

OPERATION: This subroutine first calls the EXVR subroutine to evaluate the effective address operand. Next, the USBASE subroutine is called to compute the displacement for an implied base register. If an index register is specified, the ABS8 subroutine is used to evaluate it. If a base register is specified, the ABS8 subroutine is used to evaluate it. Finally, the address alignment is checked.

EXIT: This subroutine exits to the calling point, except when a terminator error is encountered.

ERRORS: If the operand contains errors, the EXVR subroutine issues diagnostics and the effective address is rejected.

If the effective address cannot be converted into a displacement, the USBASE subroutine issues a diagnostic and the displacement is not assembled.

If either register specification is in error, the ABS8 subroutine issues a diagnostic and the specification is not used.

If the implied base register is not register 0 and a base is specified, diagnostic 40 is issued and the implied base is used (specification is ignored).

If the register(s) specification is not properly terminated, diagnostic 14 is issued and the subroutine exits to PRIVBK in the new statement routine.

If the effective address is misaligned, diagnostic 39 is issued.

3.1.2.16 Evaluate Storage Address with 4-Bit Length Subroutine (SLA4Q)

FUNCTION: This subroutine (Chart PC) evaluates a storage address with an implied or explicit length and with an implied or explicit base register specification.

ENTRY: This subroutine is entered at SLA4Q from the machine operation routine or from the SA4Z subroutine

OPERATION: This subroutine first calls the EXVR subroutine to evaluate the effective address, then calls the UBASE subroutine to develop the displacement and implied base (if any). If there is no explicit length, the implied length is validated and assembled. If an explicit length is supplied in the operand, the ABS8 subroutine is used to evaluate it.

If a base register specification follows the length specification or when the SLA4Q subroutine is called by SA4Z, the explicit base register is evaluated by ABS8. Finally, the effective address alignment is tested.

EXIT: This subroutine exits to the calling point.

ERRORS: If the operand contains errors, the EXVR subroutine issues diagnostics and the effective address (including base and displacement) is ignored.

If the effective address cannot be converted into a displacement, the USBASE subroutine issues a diagnostic and the displacement is not assembled.

If the implied length is too large, diagnostic 41 is issued, assembly of the length is bypassed, and the effective address alignment is not checked.

If the ABS8 subroutine finds an error either in the explicit length or explicit base register, it issues a diagnostic and its error return bypasses use of the term in error.

If the explicit length is void (blank) or not properly terminated, the implicit length is used in its place. (An error in the explicit length, found by ABS8, blocks use of either length.)

If an explicit base register is specified and the implicit base register is not register 0, diagnostic 40 is issued and the implied base is used.

If either the explicit length or the explicit base register is not properly terminated, diagnostic 14 is issued.

If the effective address is not properly aligned, diagnostic 39 is issued.

3.1.2.17 Evaluate Storage Address with 8-Bit Length Subroutine (SLA)

FUNCTION: This subroutine (Chart PD) evaluates a storage address with an implied or explicit length and an implied or explicit base register specification.

ENTRY: This subroutine is entered at SLA from the machine operation routine or from the SI8Z subroutine.

OPERATION: This subroutine first calls the EXVR subroutine to evaluate the effective address, then calls the USBASE subroutine to develop the displacement and implied base (if any). If no explicit length is given, the implied length is validated and assembled. If an explicit length is given in the operand, the ABS8 subroutine is called to evaluate it. If the opcode is MVW and the length is implicit, it is divided by four to give a word length instead of a byte length.

If a base register specification follows the length specification or when the SLA subroutine is called by SI8Z, the explicit base register is evaluated by ABS8. Finally, the effective address alignment is checked.

EXIT: This subroutine exits to the calling point, except when a terminator error is found.

ERRORS: If the operand contains errors, the EXVR subroutine issues diagnostics and the effective address (including base and displacement) is ignored.

If the effective address cannot be converted into a displacement, the USBASE subroutine issues a diagnostic and the displacement is not assembled.

If the implied length is too large, diagnostic 41 is issued, assembly of the length is bypassed, and the effective address alignment is not checked.

If the ABS8 subroutine finds an error either in the explicit length or explicit base register, it issues a diagnostic and its error return bypasses use of the term in error.

If the explicit length is void (blank or not properly terminated, the implicit length is used in its place. (An error in the explicit length, found by ABS8, blocks use of either length.)

If an explicit base register is specified and the implicit base register is not register 0, diagnostic 40 is issued and the implied base is used.

If either the explicit length or the explicit base register is not properly terminated, diagnostic 14 is issued and exit is made to PRIVBK in the new statement routine.

If the effective address is not properly aligned, diagnostic 39 is issued.

3.1.2.18 Evaluate Storage Address with 8-Bit Length = 0 (SI8Z)

FUNCTION: This subroutine (Chart PE) evaluates a storage address in the SI format with the immediate byte (length) containing all zeros.

ENTRY: This subroutine is entered at SI8Z from the machine operation routine.

OPERATION: This subroutine uses the SLA subroutine to evaluate the storage address, then validates the length, the assembled base, and the register specification.

EXIT: This subroutine exits to the calling point.

ERRORS: If the immediate byte is not assembled as all zeros and either the assembled base is not zero or both R1 and R2 are not zero, diagnostic 4 is issued.

3.1.2.19 Evaluate Storage Address with 4-Bit Length = 0 Subroutine (SZ4Z)

FUNCTION: This subroutine (Chart PE) evaluates a storage address with a length half-byte of all zeros.

ENTRY: This subroutine is entered at SA4Z from the machine operation routine.

OPERATION: This subroutine uses the SLA4Q subroutine to evaluate the storage address, then validates the length field.

EXIT: This subroutine exits to the calling point.

ERRORS: If the length is not assembled as all zeros, diagnostic 40 is issued.

3.1.2.20 Stack A Diagnostic Routine (DIAG)

FUNCTION: This routine (Chart RA) is used by BLPAS2 to place a diagnostic code and statement symbol in a stack of diagnostics applying to that statement. The BLPRINT routine prints, after the statement, the corresponding diagnostic message for each code in the stack.

ENTRY: This routine is entered at DIAG from BLPAS2.

EXIT: This routine returns to the calling point in BLPAS2.

3.2 GET A STATEMENT DURING BLAPS2 ROUTINE (BLIOGET2)

FUNCTION: This routine (Chart QA) is used by BLPAS2 and BLANALY to get a statement from the intermediate buffer, until it is emptied. Thereafter, the routine gets a statement from .WORK2 if it is available.

ENTRY: This routine is entered at BLIOGET2 by BLPAS2 or BLANALY.

OPERATION: This routine first checks whether the intermediate buffer has been emptied. If not, the record to be delivered is examined to determine its length and the pointer to the next record. Once the buffer is emptied, this routine reads a record from .WORK2 each time it is called.

EXIT: This routine exits to the calling point in BLPAS2 or BLANALY.

3.3 EVALUATE CONSTANT (BLCONVAL)

Routine BLCONVAL evaluates constants, using as input the modifiers (multiplicity, type, length, etc.) developed by BLCONMOD. The results of BLCONVAL operations are stored in an output table that gives the address of the assembled constant, its length, and diagnostic information. (The BLCONVAL output format is described in Subsection 6.3.) BLCONVAL also builds the Relocation List Dictionary (RLD) table.

Routine BLCONVAL has five major parts, each identified by its initial entry point, which are clearly marked in the assembly listing. The names and functions of these are as follows.

1. BLCONVAL Mainline - This directs BLCONVAL operations and processes decimal, hexadecimal, and character constants.

2. BSUBRT Routine - This evaluates fixed and floating-point constants.

3. ASUBRT Routine - This processes address constants.

4. USBAS Subroutine - This finds base and displacement for an S-type address constant.

5. QRLD Subroutine - This makes an entry in the Relocation List Dictionary table for a relocatable A-type address constant.

3.3.1 BLCONVAL Mainline

FUNCTION: This mainline (Charts KA, KB, KC, KD) initiates BLCONVAL operations, selects the routine or subroutine required to process the input constant, and converts decimal, hexadecimal, and character type constants.

ENTRY: This mainline has two entries as follows.

1. Location BLCONVAL, entered from the BLPAS2 program. The BLPAS2 program supplies the BLCONMOD output table address in a general register.

2. Location BACK, used by the BSUBRT routine at the end of processing.

OPERATION: After initialization, the mainline tests for type of constant and branches to the BSUBRT routine for a fixed- or floating-point type or to the ASUBRT routine for an address constant. Other types are handled within the mainline.

For a decimal (P or Z) type, the mainline establishes the sign, scans for a decimal point, and moves numeric characters to a work area until the end of the value list is read. Invalid characters are replaced with a zero, and a single diagnostic is stored in the BLCONVAL output. In Z-type conversion, the number of characters in the work area is then compared to the explicit length given by BLCONMOD, and the start address of the character string is modified by any difference. High-order characters are truncated and a diagnostic is set if the length is less than the number of characters; if the length is greater, the field is padded with high-order zeros. Finally, the required number of characters is moved to the BLCONVAL output table. In P-type conversion, the number of positions needed to allocate the characters in the work area is compared to the explicit length, and the start address of the character string is modified if truncation or padding is required.

Location DFIN (Chart KB) marks the end of processing for Z-, P-, and X-type constants. Here, explicit length is stored in the output table as the total length of the constant. Next, mainline checks whether any odd bits were specified and, if so, tests for bit truncation. A diagnostic is stored and excess bits are zeroed if significant bits were lost. Finally, the error count is saved before return is made to BLPAS2.

In C-type conversion (Chart KC), characters are stored unchanged in the work area until end of input is detected. After testing explicit length, the mainline moves the specified number of characters to the output table, truncating or padding if necessary. (Padding in this case means supplying trailing blanks.) The mainline then tests for odd bits, as described before, branching to the exit point.

In X-type converions (Chart KD), hexadecimal characters are moved to the work area until the end of the value list is found. Hexadecimal characters A through F are translated before being stored, invalid characters are replaced with a zero and a single diagnostic is set. The bytes needed to store all characters in the work area are compared with the specified byte length, and padding or truncation is performed accordingly. When there are more than 15 characters, the output table is packed in two or three steps before the segment branches to location DFIN for odd bit checking.

EXIT: The mainline can transfer control to any of three locations.

1. At the end of processing, return is made to the calling program. This is the only exit from the BLCONVAL program.

2. Location ASUBRT, in the ASUBRT routine, used for an A- or S-type constant.

3. Location BSUBRT, in the BSUBRT routine, used for an F-, H-, D-, or E-type constant.

ERRORS: Mainline may initiate the following diagnostic messages.

INVALID CHARACTER FOUND

VALUE SPECIFICATION MISSING - POSSIBLE ERRORS
(no numeric value supplied for a P- or Z-type constant)

TRUNCATION OF CONSTANT - POSSIBLE ERROR

3.3.2 BSUBRT Routine

FUNCTION: This routine (Charts KE, KF, KG, KH, KI) evaluates F and H (fixed-point) and E and D (floating-point) constants. Provided no error is detected, these constants are set up for conversion by the FROMF or FROMD routine and the resulting value is stored in the BLCONVAL output table.

ENTRY: The routine has three entry points.

1. Locate BSUBRT, entered from the BLCONVAL mainline when an F-, H-, D-, or E-type constant is discovered.

2. Location BTEND1, entered from the ABUBRT routine after address constant processing to check whether the entire value list has been handled.

3. Location BENDRT, entered from the ASUBRT routine when no further value list processing is desired.

OPERATION: The routine begins with a syntactical scan of the input to ready a constant for either the FROMF or FROMD routine. A sign is set and numeric characters are moved to a work area, which is arranged as the output table for a call to FROMF or FROMD. (The format of this table is described in Subsection 6.3.) When a decimal point appears, the character count is also stored in the work area to show the number of integer digits. An E results in a branch to BEXP1 (Chart KF) to process an explicit exponent. There, either the BLEXVAL routine is called to compute the value of the expression given for the exponent, or a sign is established and a scan is made for no more than two significant digits which are converted to binary and saved. Finally, after exponent conversion or upon encountering the end of the value list, the routine completes the work area/output table. It stores a terminator (a quote) after the last character if there are fewer than 23 characters; stores the total character count if no decimal point appeared; and also stores any exponent and scale factor. A branch is then taken for either fixed- or floating-point operations. (Any error detected during this set-up portion results in a branch to store a diagnostic code in the BLCONVAL output table and to end processing of the constant.)

For a floating-point constant, the routine calls FROMD (Chart KG) and tests its return. If no error is indicated, the constant value supplied by FROMD is truncated to specified size, if necessary, and rounded at the rightmost bit. Next, this result is tested to ensure the rounding did not cause a carry-over to the exponent or, if scaling was involved, to the vacated portion. If carry-over did occur, the constant is adjusted and realigned before the result is moved to the BLCONVAL output table. The routine then branches to location BTEND1 (Chart KH) to see whether the value list contains more constants.

For a fixed-point constant, bit length is added to the parameters in the work area before the FROMF call (Chart KH). Upon return, the FROMF result is checked for error indications and, if there are none, moved to the BLCONVAL output table. When an odd bit length is specified, the routine aligns the constant value before storing it.

The routine next checks (at location BTEND1, Chart KH) whether it has finished the value list. If not, it tests the next constant for type and either calls the ASUBRT routine for an A- or S-type or loops back to process the new constant. When the value list is complete, the routine passes control to the BLCONVAL mainline at the exit point, after first storing the output length and error count in the output table.

EXIT: This routine may exit to either of two locations.

1. Location BACK, in the BLCONVAL mainline, taken when the value list has been finished. BACK is the only exit point from the BLCONVAL program.

2. Location ASUBRT, in the ASUBRT routine, taken when the next constant in the value list is an A- or S-type.

ERRORS: The BSUBRT routine generates diagnostic codes for the following messages.

```
*****INVALID CHARACTER FOUND

*****FIELD n HAS A SYMBOL OR NUMBER WHICH IS TOO LONG

*****FIELD n HAS A RELOCATABLE IN PLACE OF ABSOLUTE

*****INCOMPATIBLE SCALING - POSSIBLE ERROR

*****CONSTANT HAS BEEN ROUNDED AND TRUNCATED

*****EXPONENT IS INVALID

*****FIELD n RESULTED IN A CONSTANT WHICH WAS TOO LARGE
- POSSIBLE ERROR

*****FRACTION PART LOST - POSSIBLE ERROR

*****FLOATING POINT EXPONENT UNDERFLOW - POSSIBLE ERROR
```

The routine also moves to the BLCONVAL output table any diagnostic codes generated by the BLEXVAL program; for these messages, see the BLEXVAL description.

3.3.3 ASUBRT Routine

FUNCTION: This routine (Charts KJ, KK) evaluates address (A- and S-type) constants.

ENTRY: There are two entry points to this routine.

1. Location ASUBRT, entered from the BLCONVAL mainline when an A- or S-type constant is encountered.

2. Location ARES, entered from the BTEND1 sequence of the BSUBRT routine when a subsequent expression in a multiple value list of an A- or S-type constant is found.

OPERATION: Any value handled by this routine is either an expression or a literal. For the first, the routine calls the BLEXVAL program to develop the expression value. For the other, it calls the BLSLKUP routine to get the address of the literal, using as input a dummy label stored in the assembler communications region. The return is then used to build a dummy BLEXVAL output table so that processing by the QRLD or USBAS subroutine, called later, will be uniform. If a literal is discovered but none has been indicated for the statement, the routine issues a diagnostic and passes control, via BSUBRT, to the BLCONVAL exit point.

Once a valid value is obtained, the constant type determines subsequent operations.

For an S-type constant, the USBAS subroutine is called to break the value into a base and displacement. The result, always a halfword, is then put in the BLCONVAL output table. If no usable base register is available, USBAS will generate a diagnostic message.

For an A-type constant, the routine checks whether the specified length is less than a fullword. If so, it aligns the constant value as specified and, if truncation of a significant bit occurs, sets a diagnostic code. The value is then stored in the BLCONVAL output table before the QRLD subroutine is called to build the RLD Table.

For either type, multiplicity is checked for greater than one. If this is found, the location counter is advanced by the length of the constant and the routine loops back and re-evaluates the constant at the new location to insure proper value if ''*'' is involved. The multiplicity is reduced by one on each iteration until satisfied.

Since no value is acceptable after a literal, exit from the routine depends on whether a literal or an expression was processed.

EXIT: The routine can exit to either of two locations, both within the BSUBRT routine.

1. Location BTEND1, used to check whether the value list has been completed. This exit is taken after expression handling.

2. Location BENDRT, taken to set up for an exit from BLCONVAL. This is used after literal processing.

ERRORS: The routine can generate diagnostic codes for the following messages.

FIELD n HAS BEEN TRUNCATED - POSSIBLE ERROR

Issued if a significant bit is lost when an A-type constant is aligned to specified length.

ERROR IN VALUE LIST

Issued if a character other than a right parenthesis follows a literal.

FIELD n HAS AN ERROR IN LITERAL DEFINITION

Issued when a literal is found but no literal has been indicated for the input parameter.

FIELD n RESULTED IN A CONSTANT WHICH WAS TOO LARGE - POSSIBLE ERROR

Issued if an expression value exceeds 24 bits.

The routine also moves to the BLCONVAL output table any diagnostic codes generated by the BLEXVAL program. For those messages, see the BLEXVAL description.

3.3.4 USBAS Subroutine

FUNCTION: This subroutine (Chart KL) searches the assembler USING table to develop base and displacement for an S-type address constant.

ENTRY: The only entry, location USBAS, is entered from the ASUBRT routine.

OPERATION: Using the value supplied in the BLEXVAL output table format by the ASUBRT routine, USBAS scans the entire USING table to find the corresponding base register and displacement value. (The format of the USING table appears in Subsection 6.1.) If several base registers have usable values, the one offering least displacement is selected. If displacement is the same with several registers, the register first encountered is chosen.

When no base register covers the input value, the subroutine issues a diagnostic through a call to the DIAG routine before returning control.

EXIT: Return is to the calling point.

ERRORS: The subroutine sets up the following diagnostic message if a valid base register value cannot be found.

FIELD n HAS ADDRESS WHICH IS NOT COVERED BY A USING

3.3.5 QRLD Subroutine

FUNCTION: This routine (Chart KM) makes an entry in the RLD table for a relocatable address constant.

ENTRY: Location QRLD is entered from the ASUBRT routine as part of A-type constant processing.

OPERATION: The subroutine begins by aligning the constant on a word boundary, if specified by the alignment code given in the BLCONVAL input. For a relocatable expression, the subroutine moves the contant's location, length, sign, and relocation ID number, along with the relocation ID of the current control section, to the RLD table. (The format of this table is given in Subsection 6.1.) A check is then made for table overflow before the subroutine adds the length of the constant to a dummy location counter value and returns control.

If the expression proves absolute, the subroutine merely increases the location counter value before exiting.

EXIT: Return is to the calling point.

ERRORS: The subroutine sets up the following message for the DIAG routine if a table overflow is found.

RLD TABLE OVERFLOWED

3.4 CONVERT TO FIXED POINT ROUTINE (FROMF)

FUNCTION: This routine (Charts LA, LB, LC, LD, LE) converts a decimal number, containing 1 to 23 digits in EBCDIC, to a doubleword, fixed-point, binary representation. The maximum value that can be represented is 9,223,372,036,854,775,807; the minimum is .1084203 x 10^{-18}. If negative scaling is specified, the maximum value that can be converted is +9,999,999,999,999,999,999.

ENTRY: This routine is entered at location FROMF from BLCONVAL. BLCONVAL supplies in a special data format (labeled TABLEA) the input number, exponent and length information, and scale factor. (The format of TABLEA is detailed in Subsection 6.3.4)

OPERATION: After calculating and validating the input number length and true exponent, the routine processes the integer and fraction parts of the number separately. Large segments of the routine are skipped if the input contains only an integer or fraction. Both of these number parts are handled in portions, each of which is assumed to be a right-justified integer containing a specific number of digits and an exponent. After the portions have been multiplied (or, for fraction portions, divided) by the power of 10 indicated by their exponent, they are added together to form the integer and fraction binary values. Finally, these two values are aligned according to the specifications given in TABLEA and added to obtain the fixed-point representation.

Integer processing is done in a series of loops. One loop (starting at location CL4, Chart LA) divides the integer into portions of 9 characters or less and converts them to binary. Another loop (starting at location LM6, Chart LB) multiplies each portion by 10 raised to the power of the portion exponent. The maximum exponent for the first integer portion is 10; the second portion can have an exponent of 1 or 0; the third portion, if present, contains only one digit with a 0 exponent. Multiplication is done by shifting and adding so that a multiplicand larger than 31 bits can be used without exceeding the doubleword limit. But this is not true if the scale factor is minus. So if an overflow occurs and the scale factor is minus, the routine tries to save the integer output by shifting the binary portions one bit to the right and by decreasing the scale factor. Carries from the higher-value portions are saved in the next lower portion so that the bit actually shifted out is in the lowest portion and has the least significance. If a nonzero bit is lost, the truncation flag is set, but processing still continues by restarting the multiplication loop with the new portion values.

After multiplication, the portions are added together to form the integer binary output. Overflow may occur at this time becuase of negative scaling or because the sum equals the maximum negative value. If the latter is the case, the maximum negative number is stored for output. If scaling caused the overflow, the sum of the portions is right-shifted until the scale factor is reduced to zero. The truncation flag will be set if a nonzero bit is shifted out.

If the input number contains fractional digits, the routine uses several loops to develop the fraction binary output. Three loops (beginning at locations CLI9, CKI10, and CLI101; Chart LB) determine the exponent for the fraction. If an exponent is furnished, it is complemented to positive for processing, but is understood to be negative. Therefore, it is incremented for each zero and digit found in a left-to-right scan, and reduced for each zero found in a right-to-left scan. (If the exponent exceeds 20 before the first significant digit is found, the fraction is too small to represent in 63 bits.) Another loop divides the fraction into 8-character portions and converts these to binary. Finally, a three-time loop handles exponentiation by dividing each portion by the corresponding power of 10. To gain maximum accuracy, a 96-bit number is used as the dividend. Each division produces a third of this number: the quotient of the first division is stored in the first 32 bits and its remainder is divided to get the next 32-bit value; the remainder of the second division is then divided to get the final 32-bit value. Shifting and rounding take place several times during this operation in order to prevent a fixed-point overflow or the loss of low-order bits. When division is completed, the fraction portions are added together, rounded, and shifted to make up the fraction part of the output.

The routine aligns the integer and fraction binary values by using the length and scale factor information provided by BLCONVAL in TABLEA. The two values are then added together to obtain the fixed-point representation of the input number. If the input sign was minus, the representation is complemented to negative before it is moved to the output area, TABLEA.

EXIT: This routine exits to BLCONVAL.

ERRORS: The routine may set any one of the following flags.

LE	Invalid length or scale factor specification. The output is zero.
IN	Lost integer; i.e., an uncorrectable overflow occurred. The output is zero.
FR	Lost fraction. The integer value, if any, is still placed in the output area and supplied to BLCONVAL.
LT	Low-order truncation of an integer; i.e., a significant bit was lost. The remainder of the constant is supplied to BLCONVAL.

Any diagnostic messages resulting from these errors are issued by BLCONVAL.

3.5 CONVERT TO FLOATING POINT ROUTINE (FROMD)

FUNCTION: This routine (Charts MA, MB) converts a decimal number given in EBCDIC notation to its double-precision floating-point representation. The maximum value that can be represented is $+.7237005577332262 11 \times 10^{76}$; the minimum value is $+.5397605346934028 \times 10^{-78}$.

ENTRY: This routine is entered at location FROMD from the BLCONVAL program, which supplies the input number (along with exponent and scale factor) in the data area TABLEA, also used for the output. (The formats are described in Subsection 6.3.)

OPERATION: After checking for a valid exponent, the routine processes the input number in portions of eight digits or less. Each portion is assumed to be an integer with its own exponent, a positive or negative power of ten. After the portion has been converted, it is multiplied by the appropriate power of ten obtained from one of two tables: TABP for a positive exponent; TABM for a negative. The entries in these tables are floating-point representations, each consisting of a hexadecimal exponent and 60 fraction bits for greater accuracy.

For a nonzero exponent between +76 and -75, the routine finds the location of the proper table entry by adding the portion exponent multiplied by 9 to the table address. The first time through, when greatest accuracy is needed, the table entry forms two multipliers: one with the entry exponent and high-order fraction bits; the other with the same exponent and the low-order bits. The floating-point representation of the portion is the sum of the two products: algebraically, N(A+B), where A and B are the multipliers and N is the portion.

The remaining portions are processed more directly, using only a single multiplier, since the saving of low-order bits for these portions would be superfluous. Each is scaled to have the same exponent as the first portion and rounded before it is added to the first portion value. The final total of the portions is the floating-point representation.

If the first portion is −76 or less, a special method is needed because the smallest power of 10 in the negative table is 10^{-75}. Two entries are therefore used at location LESS; one for 10 to the power of −75 and one for 10 to the power of the given exponent +75. The exponent of the first entry is increased by 8 to avoid underflow in the partial results. This increase is compensated for when the final result is obtained. Moreover, the first entry is split as previously explained to get two multipliers and products, which are then multiplied by the corresponding parts of the second table entry. The sum of these, after necessary adjustments, is the floating-point representation. (When the first portion exponent is less than −75, only one portion is handled, for the value of the other portions would be too small to represent.) For example, if N is the floating-point representation of the first portion and has an exponent of 10^{-78}, it equals N x 10^{-75}x10^{-3}. Let A1 and B1 represent the high- and low-order portions of the 10^{-75} entry and A2 and B2 stand for the corresponding parts of the 10^{-3} entry.

$$10^{-78}\ N = NA1A2 + \frac{NA1(16^6B2) + NA2(16^6B1) + \frac{N(16^6B1)(16^6B2)}{16^6}}{16^6}$$

The factor 16^{-6} is used to raise B1 and B2 to a higher value so that their products, which might otherwise be lost because of exponent underflow at multiplication time, are saved. Adjustments for the factor are performed in fixed-point registers by right-shifting fractions, and no significant bits are lost. At the same time, the products are scaled and rounded to prevent the loss of low-order bits when they are finally added together.

When scaling is requested, the routine proceeds to location LR12 after developing the floating-point representation. The fraction is right-shifted four bits for each unit of the scale factor, then rounded at the rightmost bit. If the rounding causes a carry-out into the vacated area, an additional four-bit shift and a corresponding exponent adjustment follow.

Before exiting, the routine places its output, along with an appropriate flag, in TABLEA. A zero in the flag field marks a successful conversion.

EXIT: This routine returns to the calling point.

ERRORS: If the routine finds an overflow or underflow during processing, it zeros the output area and sets a flag: -1 for an overflow; -2 for an underflow.

3.6 BLPRINT ROUTINE

The BLPRINT routine consists of 15 logical sections: the BLPRINT mainline and 14 subroutines. These are described separately on the following pages. The names and functions of these sections are as follows:

1. BLPRINT Mainline: This identifies the input statement and calls subroutines to prepare a print line, to print the statement in the assembly listing, and, if requested, to arrange TXT cards for punching.

2. DSECT Subroutine: This subroutine processes all types of input statements when a listing is being prepared and a DSECT is indicated.

3. DIAGX Subroutine: This subroutine prints any diagnostic messages for the input statement and sets error indicators in the assembler communcations region. The PRINT routine usually returns control to the calling program at the end of this subroutine.

4. SEQ Subroutine: This subroutine checks the sequence number of a statement when necessary, and sets a flag in the listing if an error is found.

5. INCSTA Subroutine: This subroutine generates the line number for a statement and moves it to the print area.

6. STA2PR Subroutine: This subroutine sets up a statement for printing and moves it to the print area.

7. ASMLOC Subroutine: This subroutine sets up the current location counter value and places it in the print area.

8. ASMCON Subroutine: This subroutine sets up a constant generated for alignment and places it in the print area.

9. ROUT Subroutine: This subroutine sets up a print line for a constant requiring two or more lines and moves it to the print area.

10. PRTIT Subroutine: This subroutine prints a line in the assembly listing.

11. PUN Subroutine: This subroutine handles the punching of TXT cards, storing object code in a card area until ready for punching.

12. NUCD Subroutine: This subroutine is used to fill out and punch a card during DC, DCL, and alignment statement processing.

13. PUCD Subroutine: This subroutine punches the information stored in the card area.

14. DS1 Subroutine: This subroutine handles DS statements when there is no listing. The subroutine causes the contents of the card area to be punched.

15. DS Subroutine: This subroutine prints DS statements when there is a listing and causes the contents of the card area to be punched.

3.6.1 BLPRINT Mainline

FUNCTION: Depending on the type of input statement, the mainline (Charts TA, TB, TC) calls subroutines to arrange the statement for printing and print it in the assembly listing, to set up and punch TXT cards, and to print diagnostic messages. One statement is processed on each call to BLPRINT.

ENTRY: The mainline has a primary entry point, location BLPRINT, which is entered from the BAL and BLPAS2 programs. The calling program provides the statement address and a return address in general registers.

There are two secondary entry points: location EXIT, entered from many subroutines within the BLPRINT routine when they have finished their operations; and location COMNT, entered from the DSECT subroutine during comment processing.

OPERATION: The mainline may take either of two paths, depending on whether or not a listing is currently in progress. Each path processes according to the type of input statement. However any statement with a diagnostic will be forced through the ''listing-on'' path.

When a listing has not been requested or is presently suppressed, the mainline calls the PUN subroutine to set up TXT cards for source, DCL, and alignment statements, and branches to the DS1 subroutine for DC-type statements. Ignore and comment statements are disregarded. When listing is temporarily suppressed, the mainline also calls the SEQ subroutine to check the sequence numbers of all statements except alignment and DCL.

When a listing is being prepared, the mainline first checks whether a DSECT is in progress. If so, control passes to the DSECT subroutine for all types of statements. Otherwise, the mainline processes the input statement.

A source statement results in a series of branches to subroutines to check the sequence number; set up the line number; move the statement to the print area; get the current location counter value; print the resulting line; and, if desired, set up a TXT card with the object code generated by the statement.

An ignore statement (a statement previously found in error) is assigned a line number and printed. No punching is done.

For an alignment statement (issued by the assembler to set a DC or instruction at the proper word boundary), subroutines are called to arrange and print the current location counter value and the alignment constant. Another branch is taken to punch the object code generated for the statement if there is currently a partially-filled TXT card, or force reinitialization on the next TXT card.

A comment is given a sequence check, assigned a line number if it appears after the program START card, and printed. No punching is performed.

A DCL (literal) statement is handled according to programmer request. If the printing of literals is suppressed, the PUN subroutine is called to set up a TXT card with the object code generated by the statement. Otherwise, the mainline moves a flag to the print area to signal a literal and processes the statement much like a DC.

A DC-type statement is sequence checked, assigned a line number, and moved to the print area only if its multiplicity is one. All DC-type statements are assigned the current location counter value. When more than one print line is needed (a DC may extend up to 255 bytes), subroutines are called to set up and print the constant in 16-byte increments, although only the first such line is printed if the programmer has requested no data. Finally, the object code generated by the statement is issued to the PUN subroutine. A DS statement, detected as a DC-type with zero object length, is printed and causes the DS subroutine to punch the contents of the card area.

After statement processing is completed, the mainline branches to the DIAGX subroutine to issue any diagnostic messages stored for the current statement. Final exit from the BLPRINT routine is usually made from the DIAGX subroutine. However, an alignment statement, which cannot have a diagnostic, causes the mainline to exit directly to the calling program.

EXIT: The mainline can exit to three locations, depending on the type of input statement.

1. To the DIAGX subroutine, taken to print any diagnostic messages for the current statement. DIAGX returns control to the program that called BLPRINT.

2. To the DSECT subroutine to print all statements within a DSECT.

3. To the calling program. The mainline makes this return, at EXIT, when an alignment statement was processed.

ERRORS: No error messages are issued, but the mainline does set flag D in the print line when a literal is listed. In this case, the meaning of the flag appears at the end of the listing.

A DATA CONSTANT WAS GENERATED AS A RESULT OF A LITERAL SPECIFICATION.

3.6.2 DSECT Subroutine

FUNCTION: This subroutine (Chart TD) handles the printing of all types of statements within a DSECT (dummy control section). It operates only when a listing is to be prepared.

ENTRY: This subroutine has only one entry point, location DSECT. It is generated from the BLPRINT machine when a listing and a DSECT are both indicated.

OPERATION: The subroutine identifies the type of statement and then calls appropriate subroutines to ready and print it.

For alignment statements and for lines generated for a constant requiring multiplicity, only the current location counter value is printed. All other types of statements are sequence checked, if required; assigned a line number in the assembly listing; set up for printing; and then printed. Source, DCL, and DC-type statements are also assigned the current location counter value before printing.

EXIT: This subroutine can exit to two locations.

1. COMNT, in the BLPRINT mainline, used to set up and print a comment statement.
2. EXIT, in the BLPRINT mainline, taken when the DSECT subroutine has finished.

COMMENT: Object code within a DSECT is not punched. When there is no listing, tests within the individual BLPRINT subroutines prevent the punching of object code within a DSECT.

3.6.3 DIAGX Subroutine

FUNCTION: This subroutine (Chart TE) prints all diagnostic messages generated during assembly for the current statement. It also maintains, in the assembler communications region, counts of all serious and possible errors in the source program, and sets the fatal-error switch in the communications region when a serious error is found.

ENTRY: The subroutine has one entry point, location DIAGX. It is entered from the BLPRINT mainline after the current statement has been printed and/or punched.

OPERATION: If there is no diagnostic code in the diagnostic stack area, the subroutine immediately exits to the program that called BLPRINT.

When a code is found, the subroutine uses a loop to process all diagnostics for the current statement. If a listing has been requested, each code results in the corresponding diagnostic message being moved from an internal table to the message print area, after adjustments, if required, to include a symbol or field number. DIAGX then calls the BLLIST routine to print the message. Serious error messages are always printed when there is a listing; possible error messages may be suppressed by programmer request.

Regardless of whether printing occurs, DIAGX increments the proper error count in the assembler communications region for every code encountered. For a serious error, it also sets the fatal-error switch, thus preventing execution of the current job.

EXIT: The DIAGX subroutine exits to the calling program. This marks the end of BLPRINT routine processing.

COMMENT: BAL informs the programmer of the number of possible and serious errors by printing the count(s) even if no listing was requested. If one or more serious errors exist, it will also signal the monitor that execution of the current program is not permitted, causing the following monitor message to be printed:

PROGRAM CANNOT BE EXECUTED,

unless the ''LOAD'' option was specified.

3.6.4 SEQ Subroutine

FUNCTION: This subroutine (Chart TF) validates the number of the current statement when sequence number checking is requested. If the statement is out of sequence, SEQ moves a flag to the print area.

ENTRY: This subroutine has one entry point, location SEQ. It is entered from many points in BLPRINT.

OPERATION: The subroutine tests an indicator in the assembler communications region (MASTER) to find whether sequence checking was requested. If so, SEQ locates the sequence number of the current statement and compares it to the last sequence number. If the current number is smaller or equal, a flag is moved to the print area and an indicator is set in MASTER to indicate the errors: SEQ then stores the current number for the next comparison.

EXIT: The subroutine exits to the calling point.

ERRORS: Where the current sequence number is lower than the preceding one, SEQ inserts flag A in the print area. The meaning of the flag, defined at the end of the assembly listing is as follows:

> SEQUENCE NO. OF STATEMENT IS SMALLER THAN OR EQUAL TO PREVIOUS AND ISEQ WAS REQUESTED.

3.6.5 INCSTA Subroutine

FUNCTION: This subroutine (Chart TF) increments the line number, converts it to EBCDIC for printing, and moves it to the print area.

ENTRY: Entry to this subroutine is location INCSTA. It is entered from many points in BLPRINT.

EXIT: The subroutine exits to the calling point.

3.6.6 PRTIT Subroutine

FUNCTION: This subroutine (Chart TF) prints one line of the assembly listing each time it is called.

ENTRY: The only entry to this subroutine is location PRTIT. It is entered from many points in BLPRINT.

OPERATION: The subroutine prints the contents of the print area by callihg BLLIST. When an eject or space is needed, the subroutine also branches to the BLLIST subroutine.

3.6.7 STA2PR Subroutine

FUNCTION: This subroutine (Chart TG) sets up a statement for printing and moves it to the print area.

ENTRY: The subroutine has one entry point, location STA2PR. It is entered from the PRINT mainline and from the DSECT subroutine.

OPERATION: The subroutine moves statement fields one by one to the print area, testing the symbol, operation code, and operand fields for excessive length. For the first two fields, excessive length results merely in the readjustment of the starting location of the next field. For an excessively long operand and field, STA2PR moves a flag to the print area, sets an indicator in the assembler communications region, and then moves only 59 bytes of the operand. Columns 72-80 of the statement are moved to the print area without inspection.

EXIT: This subroutine exits to the calling point.

ERRORS: Subroutine STA2PR places flag B in the print area if an operand of excessive length is found. The meaning of this flag, defined at the end of the assembly listing is as follows:

> STATEMENT TRUNCATED ON LISTING BECAUSE THE ENTIRE OPERAND-COMMENT COULD NOT FIT ON A LINE

3.6.8 ASMLOC Subroutine

FUNCTION: This subroutine (Chart TH) converts the current location counter value from hexadecimal to EBCDIC representation and places the result in the print area.

ENTRY: The subroutine is entered at location ASMLOC from the DSECT subroutine and from several points in the BLPRINT mainline.

OPERATION: After setting up the location counter value and the proper storage area, this subroutine makes use of the loop beginning at location ASLP to convert hexadecimal characters to EBCDIC. This loop is also shared by the ASMCON and ROUT subroutines.

Conversion is accomplished by fetching two hexadecimal characters at a time, moving each to a separate byte and then operating on the bytes. Hexadecimal digits of value 9 or less are zoned with a hexadecimal F; digits A through F are reduced by 9 and zoned with a hexadecimal C. The loop starting at ASLP continues until all hexadecimal characters have been converted and stored in the designated area.

EXIT: This subroutine exits to the calling point.

3.6.9 ASMCON Subroutine

FUNCTION: This subroutine (Chart TH) converts a generated constant from hexadecimal to EBCDIC representation and places the result in the print area.

ENTRY: There is one entry point, location ASMCON. It is entered from the BLPRINT mainline when an alignment statement is being processed.

OPERATION: This subroutine obtains from the assembler communications region the constant generated during pass 2 to align a DC or an instruction on the proper word boundary. The subroutine then uses the loop beginning at location ASLP to convert the constant to EBCDIC. For a description of loop operation, see ASMLOC subroutine.

EXIT: This subroutine exits to the calling point.

3.6.10 ROUT Subroutine

FUNCTION: This subroutine (Chart TH) converts a designated portion of a DC statement from hexadecimal to EBCDIC representation and places the result in the print area.

ENTRY: The subroutine has one entry point, location ROUT. It is entered from the BLPRINT mainline when a DC statement requires more than one print line.

OPERATION: The calling routine provides, in general registers, the location of the constant value to be converted and the number of bytes involved. Usually, ROUT is requested to convert 16 bytes on each call. If the last line to be printed has fewer bytes, the exact number will be indicated by the calling routine. ROUT uses the loop beginning at location ASLP (shared with the ASMCON and ASMLOC subroutines) to convert value to EBCDIC. See the ASMLOC subroutine for a description of loop operation.

EXIT: This subroutine exits to the calling point.

3.6.11 PUN Subroutine

FUNCTION: When punching is valid, this subroutine (Chart TI) stores the object code generated by a statement on each call. The number of bytes stored in the card area determines when PUN calls the BLPUNC2 routine or the PUCD and NUCD subroutines to punch a card.

ENTRY: This subroutine has only one entry point, location PUN. It is entered from many points in BLPRINT, whenever the object code is ready for punching.

OPERATION: The subroutine begins by examining switches in the assembler communications region (in BAL) to decide whether punching is valid, exiting immediately to the BLPRINT mainline if a DSECT is indicated or if there is no punch or execute request. The subroutine then sets up a card area for the initial statement and for the next statement after a DS. (The card area is automatically punched by the DS or DS1 subroutine when a DS statement is processed; an indicator is set to inform PUN.) Statement type determines further processing.

For regular statements (see BLPRINT Mainline) the segment beginning at location PUN3 is used to store the object code in the card area, after first converting EBCDIC characters to hexadecimal. When the contents of the card area plus the code for the current statement exceed the card length, the card area contents are moved to the punch area and punched by a call to the BLPUNC2 routine. The code for the current statement is then moved to the cleared card area to begin packing of the next card.

For DCL, DC, or alignment statements, which may be much longer than a single card, the segment beginning at PUN4 is used. The PUCD subroutine is called to punch a full card only; NUCD is called to fill out the card before punching, using bytes from the current statement, and to continue punching full cards until the remaining bytes of the constant fit within a card. Subroutine NUCD then moves these remaining bytes to the card area for future punching.

Comment and ignore statements are disregarded.

EXIT: This subroutine can exit to two locations.

1. EXIT, in the BLPRINT mainline, taken after PUN completes its operations for a statement.

2. NUCD, in the NUCD subroutine, taken to fill out and punch a card during DC, DCL, and alignment statement processing. NUCD does not return to PUN, but transfers directly to EXIT in the BLPRINT mainline.

COMMENT: When an END card is encountered during pass 2, the assembler generates a DS to ensure that the code for the last statements processed is punched.

3.6.12 PUCD Subroutine

FUNCTION: This subroutine (Chart TJ) moves the contents of the card area to the punch area, calls the BLPUNC2 routine to punch a card, and reinitializes the card area. This function is performed each time PUCD is called.

ENTRY: The subroutine has one entry point, location PUCD. It is entered from the NUCD, DS, DS1, and PUN subroutines when a card is to be punched.

EXIT: This subroutine exits to the calling subroutine.

3.6.13 NUCD Subroutine

FUNCTION: This subroutine (Chart TJ) punches cards during DC, DCL, and alignment statement processing until the number of bytes of object code left in the current statement will fit within the card.

ENTRY: The only entry to this subroutine is location NUCD. It is entered from the PUN subroutine when a card is ready to be punched and a DC, DCL, or alignment statement is being processed.

OPERATION: The subroutine begins by moving to the card area enough bytes from the current statement to fill a card; it then calls PUCD for punching. Since the length of a constant may greatly exceed the length of a single card, the subroutine loops to issue full cards until the unpunched bytes of object code for the statement can be contained within a card. These are stored in the card area for future punching before the subroutine exits.

EXIT: This subroutine exits to location EXIT in the PRINT mainline.

3.6.14 DS1 Subroutine

FUNCTION: This subroutine (Chart TK) inspects a DC-type statement and, if certain conditions are met, initiates punching. DS1 operates only when a listing is not being prepared.

ENTRY: The subroutine has one entry point, location DS1. It is entered from the BLPRINT mainline when a DC-type statement is detected and there is no listing.

OPERATION: This subroutine begins by determining whether the current statement is a DC or DS. A DC statement is passed on to the PUN subroutine for possible punching. For a DS statement, the DS1 subroutine sets a switch in PUN so that the card area will be reinitialized before the next punch. It then makes a series of tests to find whether punching is permissible, calling the PUCD subroutine, when necessary, to punch the current contents of the card area.

EXIT: This subroutine can exit to two locations.

1. PUN, in the PUN subroutine, used when the statement is a DC. No return is made to DS1.

2. EXIT, in the BLPRINT mainline, taken after DS processing is finished.

3.6.15 DS Subroutine

FUNCTION: This subroutine (Chart TK) initiates printing and, when certain conditions are met, punching for a DS statement. The DS subroutine operates only when there is a listing.

ENTRY: The subroutine has only one entry point, location DS. It is entered from the BLPRINT mainline when a DS statement (DC-type statement with zero object length) has been detected.

OPERATION: This subroutine calls the PRTIT subroutine to print the DS statement in the listing, after setting a switch in the PUN subroutine so that the card area is reinitialized before the next card is punched. The subroutine then performs the same tests made by the DS1 subroutine to determine whether punching is valid. If so, PUCD is called to punch the present contents of the card area.

EXIT: This subroutine exits to location EXIT in the BLPRINT mainline.

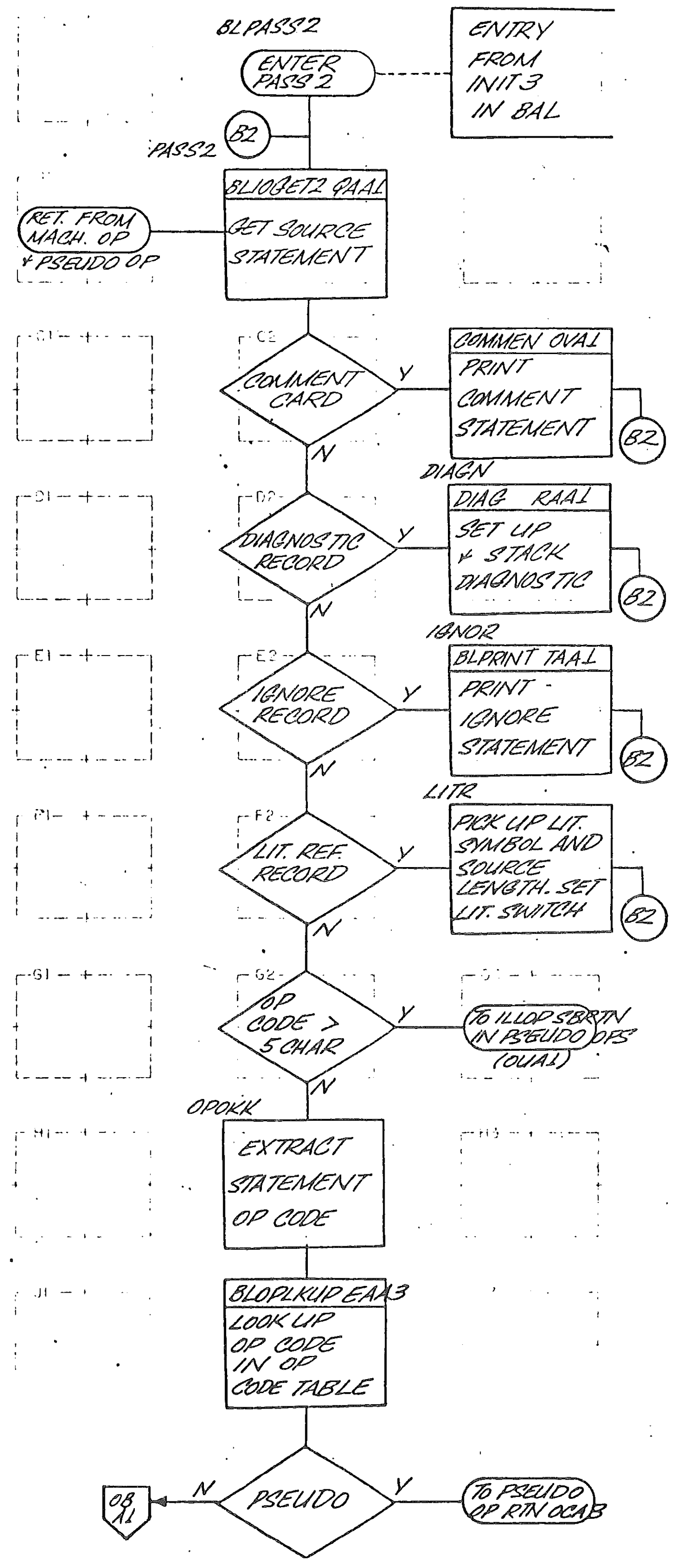
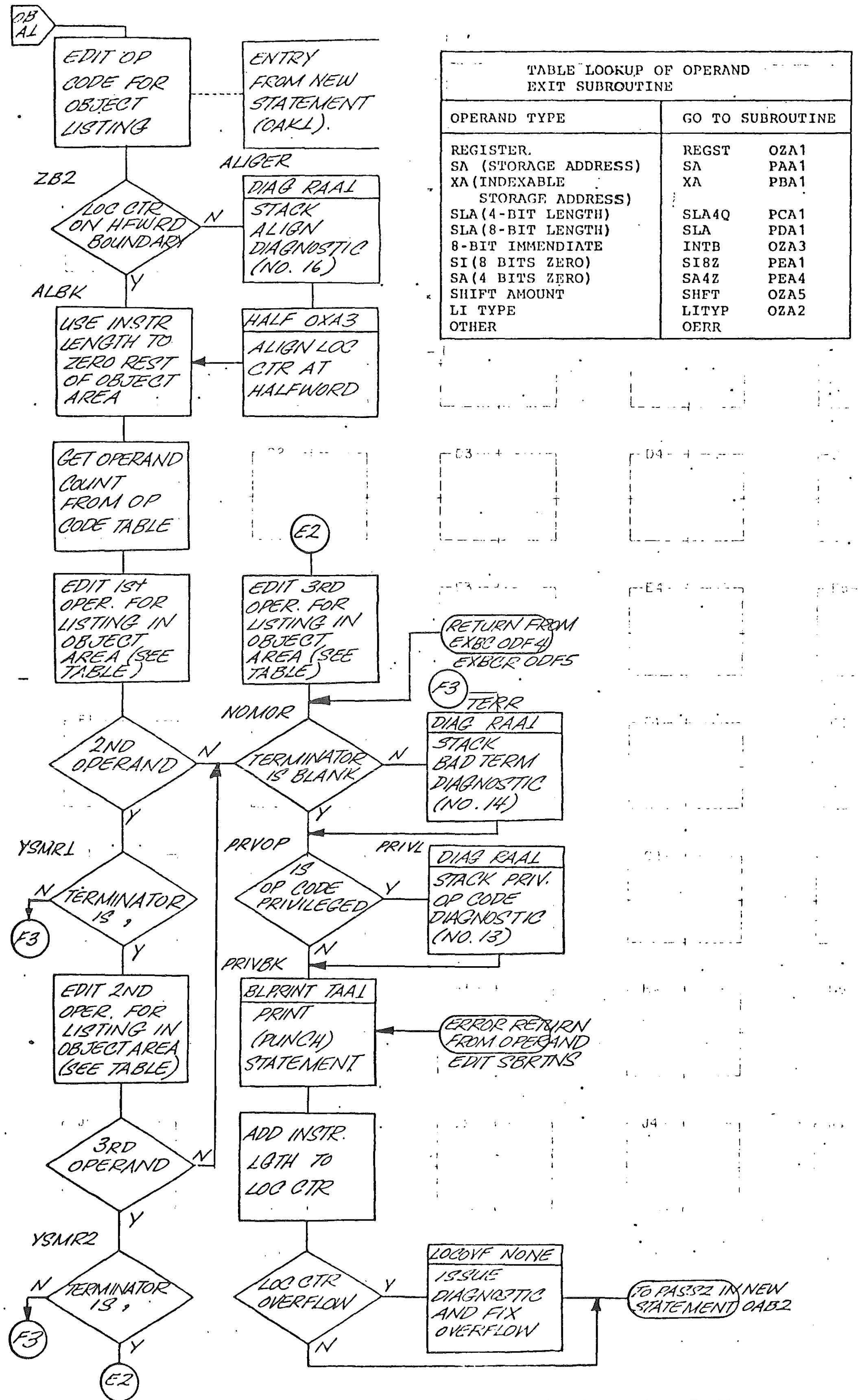


CHART OA. PASS2 ROUTINE



OB AL
EDIT OP CODE FOR OBJECT LISTING
ENTRY FROM NEW STATEMENT (OAKL).
ZB2
LOC CTR ON HFWRD BOUNDARY
N
ALIGER
DIAG RAAL
STACK ALIGN DIAGNOSTIC (NO. 16)
Y
ALBK
USE INSTR LENGTH TO ZERO REST OF OBJECT AREA
HALF OXA3
ALIGN LOC CTR AT HALFWORD
GET OPERAND COUNT FROM OP CODE TABLE
E2
EDIT 1st OPER. FOR LISTING IN OBJECT AREA (SEE TABLE)
EDIT 3RD OPER. FOR LISTING IN OBJECT AREA (SEE TABLE)
RETURN FROM EXBC ODF4
EXBCR ODF5
F3
TERR
NOMOR
2ND OPERAND
N
TERMINATOR IS BLANK
N
DIAG RAAL
STACK BAD TERM DIAGNOSTIC (NO. 14)
Y
Y
YSMR1
PRVOP
PRIVL
IS OP CODE PRIVILEGED
Y
DIAG RAAL
STACK PRIV. OP CODE DIAGNOSTIC (NO. 13)
N
TERMINATOR IS ,
F3
Y
N
PRIVBK
EDIT 2ND OPER. FOR LISTING IN OBJECT AREA (SEE TABLE)
BLPRNT TAAL
PRINT (PUNCH) STATEMENT
ERROR RETURN FROM OPERAND EDIT SBRTNS
3RD OPERAND
N
ADD INSTR. LGTH TO LOC CTR
Y
YSMR2
N
TERMINATOR IS ,
F3
Y
E2
LOC CTR OVERFLOW
Y
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
N
TO PASS2 IN NEW STATEMENT OAB2

TABLE LOOKUP OF OPERAND EXIT SUBROUTINE

OPERAND TYPE	GO TO SUBROUTINE	
REGISTER.	REGST	OZA1
SA (STORAGE ADDRESS)	SA	PAA1
XA (INDEXABLE STORAGE ADDRESS)	XA	PBA1
SLA(4-BIT LENGTH)	SLA4Q	PCA1
SLA(8-BIT LENGTH)	SLA	PDA1
8-BIT IMMENDIATE	INTB	OZA3
SI(8 BITS ZERO)	SI8Z	PEA1
SA(4 BITS ZERO)	SA4Z	PEA4
SHIFT AMOUNT	SHFT	OZA5
LI TYPE	LITYP	OZA2
OTHER	OERR	

CHART OB. MACHINE OPERATION ROUTINE

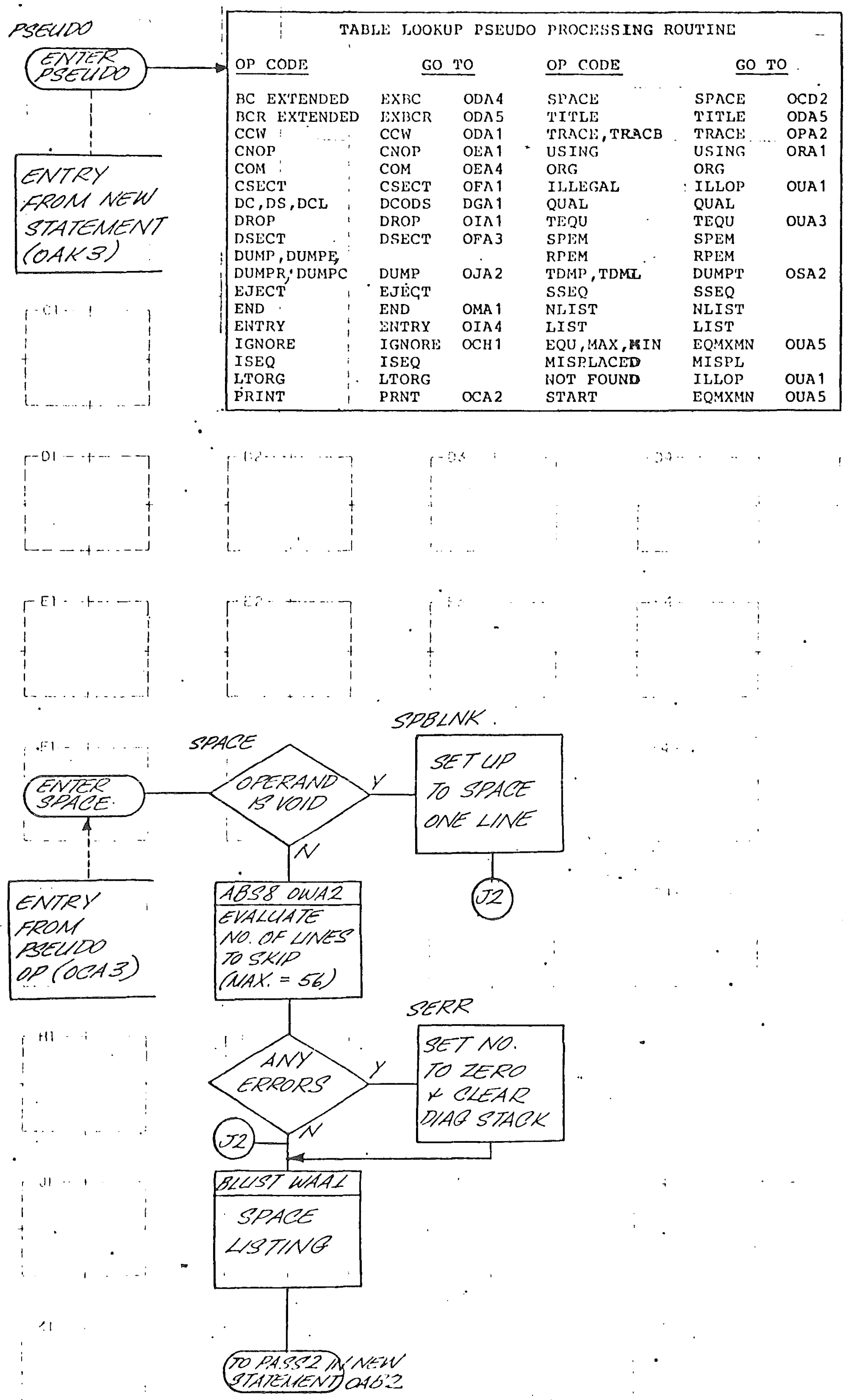

PSEUDO
ENTER PSEUDO
ENTRY FROM NEW STATEMENT (OAK3)
SPACE
ENTER SPACE
OPERAND IS VOID
Y
N
SPBLNK
SET UP TO SPACE ONE LINE
J2
ENTRY FROM PSEUDO OP (OCA3)
ABS8 OWA2
EVALUATE NO. OF LINES TO SKIP (MAX. = 56)
SERR
ANY ERRORS
Y
N
SET NO. TO ZERO + CLEAR DIAG STACK
J2
BLUST WAAL
SPACE LISTING
TO PASS2 IN NEW STATEMENT OAB2

TABLE LOOKUP PSEUDO PROCESSING ROUTINE

OP CODE	GO TO		OP CODE	GO TO	
BC EXTENDED	EXBC	ODA4	SPACE	SPACE	OCD2
BCR EXTENDED	EXBCR	ODA5	TITLE	TITLE	ODA5
CCW	CCW	ODA1	TRACE,TRACB	TRACE	OPA2
CNOP	CNOP	OEA1	USING	USING	ORA1
COM	COM	OEA4	ORG	ORG	
CSECT	CSECT	OFA1	ILLEGAL	ILLOP	OUA1
DC,DS,DCL	DCODS	DGA1	QUAL	QUAL	
DROP	DROP	OIA1	TEQU	TEQU	OUA3
DSECT	DSECT	OFA3	SPEM	SPEM	
DUMP,DUMPE,			RPEM	RPEM	
DUMPR,DUMPC	DUMP	OJA2	TDMP,TDML	DUMPT	OSA2
EJECT	EJECT		SSEQ	SSEQ	
END	END	OMA1	NLIST	NLIST	
ENTRY	ENTRY	OIA4	LIST	LIST	
IGNORE	IGNORE	OCH1	EQU,MAX,MIN	EQMXMN	OUA5
ISEQ	ISEQ		MISPLACED	MISPL	
LTORG	LTORG		NOT FOUND	ILLOP	OUA1
PRINT	PRNT	OCA2	START	EQMXMN	OUA5

CHART OC. PSEUDO ROUTINE

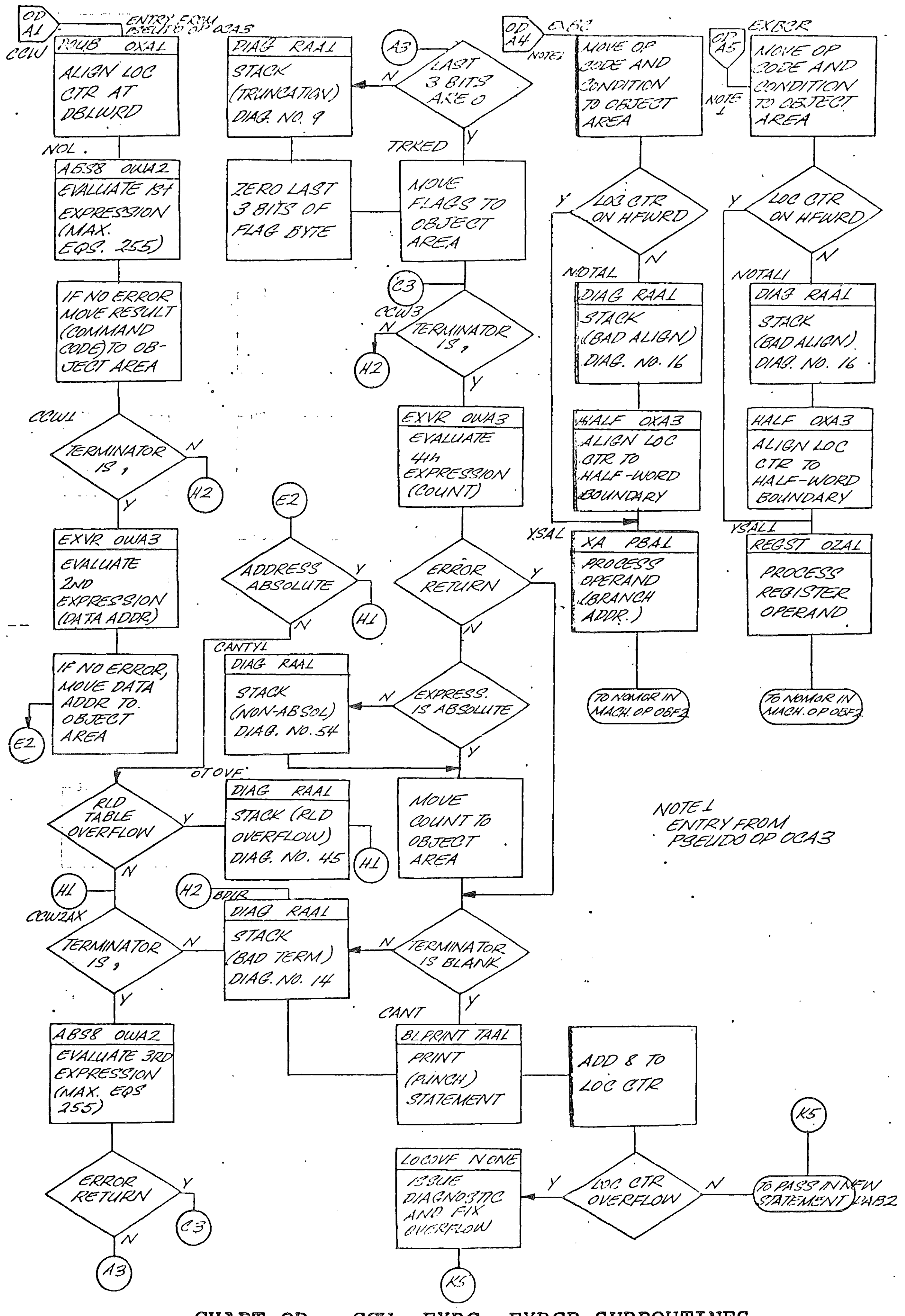

OD A1
ENTRY FROM PSEUDO OP OCA5
CCW
ROUB OXAL
ALIGN LOC CTR AT DBLWRD
NOL
ABS8 OWA2
EVALUATE 1st EXPRESSION (MAX. EQS. 255)
IF NO ERROR MOVE RESULT (COMMAND CODE) TO OBJECT AREA
CCW1
TERMINATOR IS ,
H2
EXVR OWA3
EVALUATE 2ND EXPRESSION (DATA ADDR)
IF NO ERROR, MOVE DATA ADDR TO OBJECT AREA
E2
DIAG RAAL
STACK (TRUNCATION) DIAG. NO. 9
ZERO LAST 3 BITS OF FLAG BYTE
A3
LAST 3 BITS ARE 0
TRKED
MOVE FLAGS TO OBJECT AREA
C3
CCW3
TERMINATOR IS ,
EXVR OWA3
EVALUATE 4th EXPRESSION (COUNT)
ADDRESS ABSOLUTE
H1
ERROR RETURN
CANTYL
DIAG RAAL
STACK (NON-ABSOL) DIAG. NO. 54
EXPRESS. IS ABSOLUTE
OTOVF
RLD TABLE OVERFLOW
DIAG RAAL
STACK (RLD OVERFLOW) DIAG. NO. 45
MOVE COUNT TO OBJECT AREA
CCW2AX
TERMINATOR IS ,
BPIR
DIAG RAAL
STACK (BAD TERM) DIAG. NO. 14
TERMINATOR IS BLANK
CANT
ABS8 OWA2
EVALUATE 3RD EXPRESSION (MAX. EQS 255)
ERROR RETURN
BLPRINT TAAL
PRINT (PUNCH) STATEMENT
ADD 8 TO LOC CTR
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
K5
LOC CTR OVERFLOW
TO PASS IN NEW STATEMENT PAB2
OD A4
EXBC
NOTE1
MOVE OP CODE AND CONDITION TO OBJECT AREA
LOC CTR ON HFWRD
NOTAL
DIAG RAAL
STACK (BAD ALIGN) DIAG. NO. 16
HALF OXA3
ALIGN LOC CTR TO HALF-WORD BOUNDARY
YSAL
XA PBAL
PROCESS OPERAND (BRANCH ADDR.)
TO NOMOR IN MACH. OP OBF2
OD A5
EXBCR
NOTE 1
MOVE OP CODE AND CONDITION TO OBJECT AREA
LOC CTR ON HFWRD
NOTAL1
DIAG RAAL
STACK (BAD ALIGN) DIAG. NO. 16
HALF OXA3
ALIGN LOC CTR TO HALF-WORD BOUNDARY
YSAL1
REGST OZAL
PROCESS REGISTER OPERAND
TO NOMOR IN MACH. OP OBF2
NOTE 1
ENTRY FROM PSEUDO OP OCA3
Y
N

CHART OD. CCW, EXBC, EXBCR SUBROUTINES

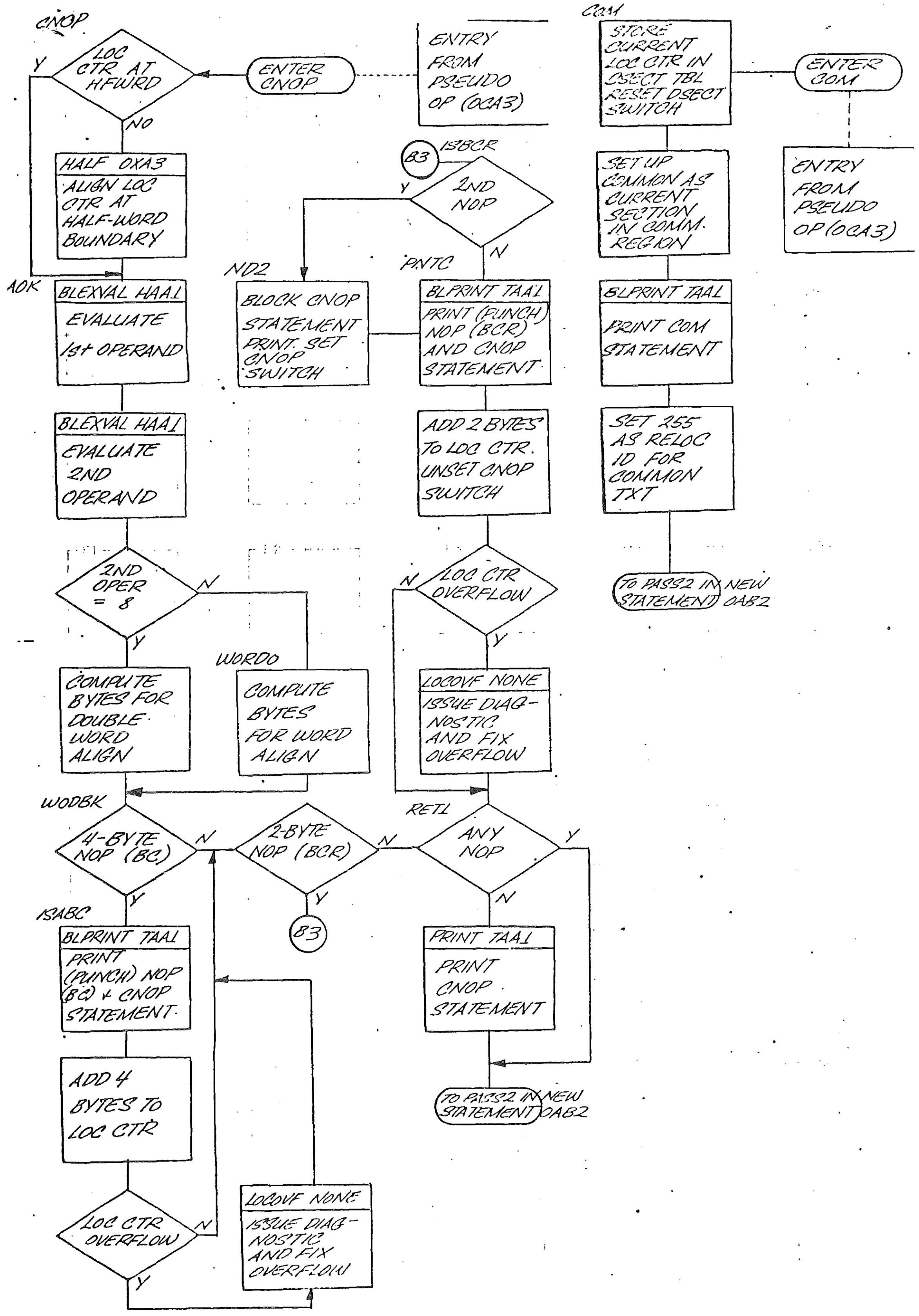

CNOP
LOC CTR AT HFWRD
Y
NO
ENTER CNOP
ENTRY FROM PSEUDO OP (OCA3)
HALF OXA3
ALIGN LOC CTR AT HALF-WORD BOUNDARY
AOK
BLEXVAL HAA1
EVALUATE 1st OPERAND
BLEXVAL HAA1
EVALUATE 2ND OPERAND
2ND OPER = 8
N
Y
WORDO
COMPUTE BYTES FOR DOUBLE WORD ALIGN
COMPUTE BYTES FOR WORD ALIGN
WODBK
4-BYTE NOP (BC)
N
Y
ISABC
BLPRINT TAA1
PRINT (PUNCH) NOP (BC) + CNOP STATEMENT
ADD 4 BYTES TO LOC CTR
LOC CTR OVERFLOW
N
Y
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
B3
ISBCR
2ND NOP
Y
N
ND2
BLOCK CNOP STATEMENT PRINT. SET CNOP SWITCH
PNTC
BLPRINT TAA1
PRINT (PUNCH) NOP (BCR) AND CNOP STATEMENT
ADD 2 BYTES TO LOC CTR. UNSET CNOP SWITCH
LOC CTR OVERFLOW
N
Y
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
2-BYTE NOP (BCR)
N
Y
B3
RETL
ANY NOP
Y
N
PRINT TAA1
PRINT CNOP STATEMENT
TO PASS2 IN NEW STATEMENT OAB2
COM
STORE CURRENT LOC CTR IN DSECT TBL RESET DSECT SWITCH
ENTER COM
ENTRY FROM PSEUDO OP (OCA3)
SET UP COMMON AS CURRENT SECTION IN COMM. REGION
BLPRINT TAA1
PRINT COM STATEMENT
SET 255 AS RELOC ID FOR COMMON TXT
TO PASS2 IN NEW STATEMENT OAB2

CHART OE. CNOP, COM SUBROUTINES

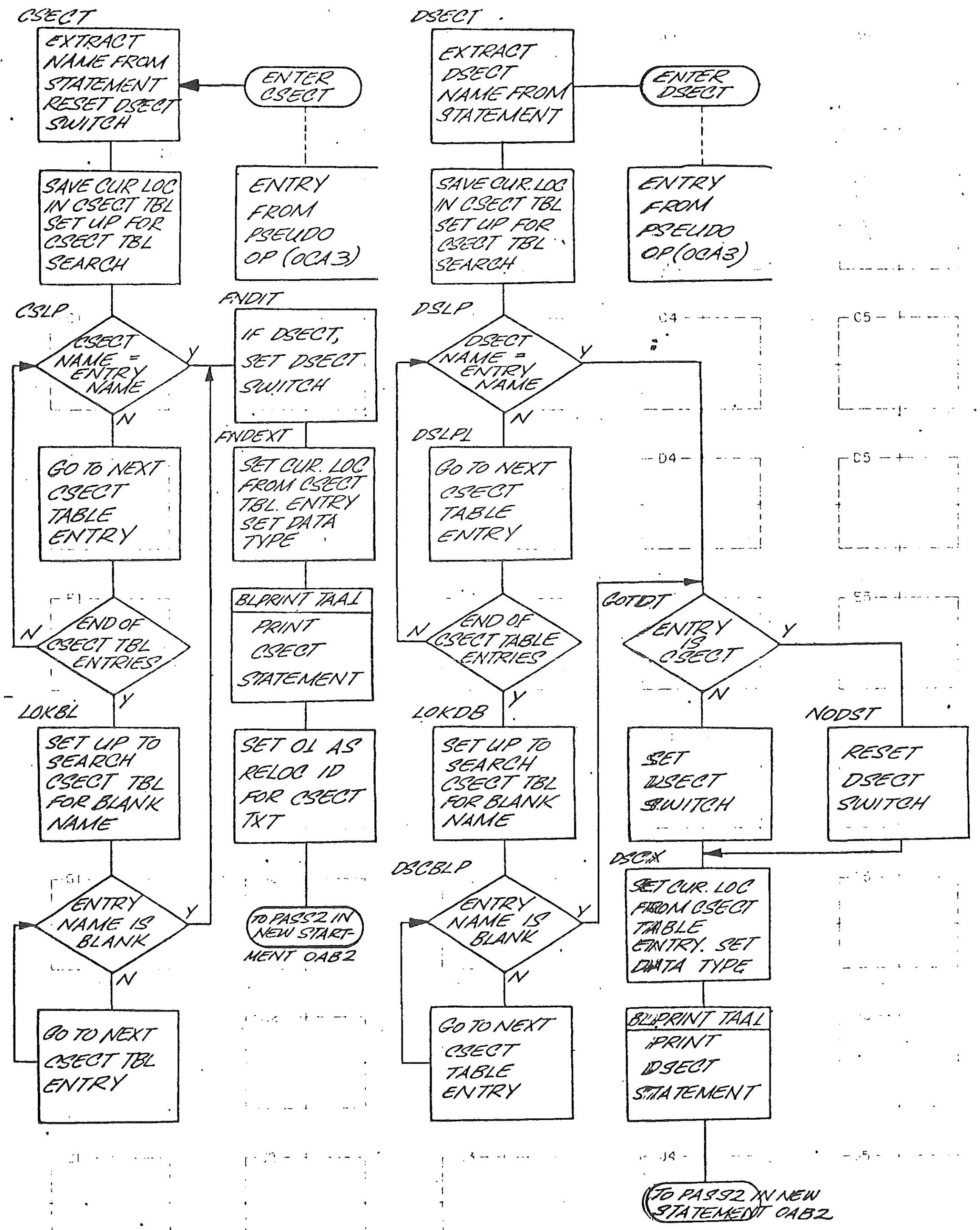

CSECT
EXTRACT NAME FROM STATEMENT RESET DSECT SWITCH
ENTER CSECT
SAVE CUR LOC IN CSECT TBL SET UP FOR CSECT TBL SEARCH
ENTRY FROM PSEUDO OP (OCA3)
CSLP
CSECT NAME = ENTRY NAME
Y
N
FNDIT
IF DSECT, SET DSECT SWITCH
FNDEXT
SET CUR. LOC FROM CSECT TBL. ENTRY SET DATA TYPE
GO TO NEXT CSECT TABLE ENTRY
END OF CSECT TBL ENTRIES
BLPRINT TAAL
PRINT CSECT STATEMENT
LOKBL
SET UP TO SEARCH CSECT TBL FOR BLANK NAME
SET 01 AS RELOC ID FOR CSECT TXT
ENTRY NAME IS BLANK
TO PASS2 IN NEW STATEMENT OAB2
GO TO NEXT CSECT TBL ENTRY
DSECT
EXTRACT DSECT NAME FROM STATEMENT
ENTER DSECT
SAVE CUR LOC IN CSECT TBL SET UP FOR CSECT TBL SEARCH
ENTRY FROM PSEUDO OP(OCA3)
DSLP
DSECT NAME = ENTRY NAME
DSLPL
GO TO NEXT CSECT TABLE ENTRY
END OF CSECT TABLE ENTRIES
GOTDT
ENTRY IS CSECT
LOKDB
SET UP TO SEARCH CSECT TBL FOR BLANK NAME
NODST
SET DSECT SWITCH
RESET DSECT SWITCH
DSCBLP
DSCX
ENTRY NAME IS BLANK
SET CUR. LOC FROM CSECT TABLE ENTRY. SET DATA TYPE
GO TO NEXT CSECT TABLE ENTRY
BLPRINT TAAL
PRINT DSECT STATEMENT
TO PASS2 IN NEW STATEMENT OAB2

CHART OF. CSECT, DSECT SUBROUTINES

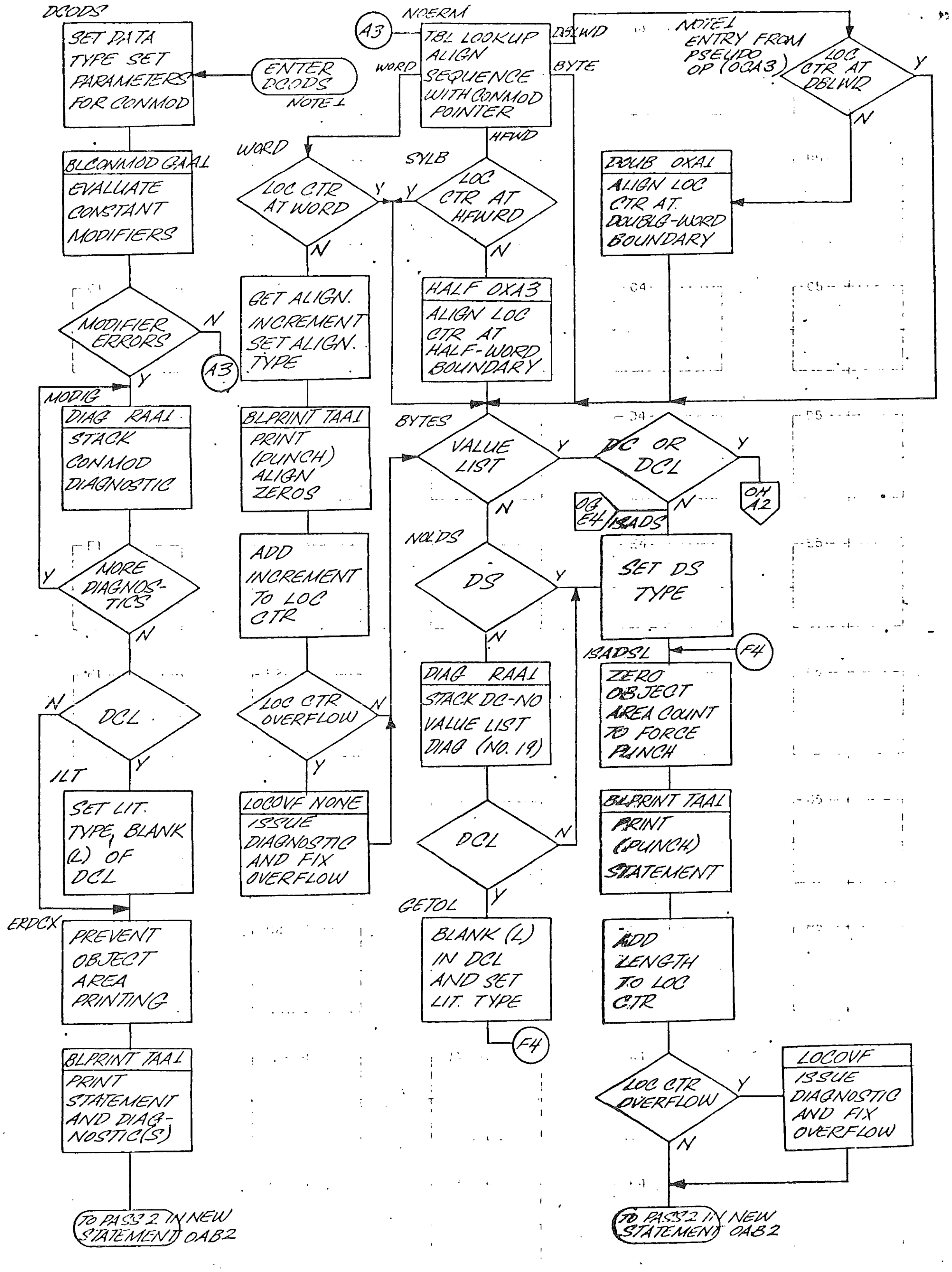
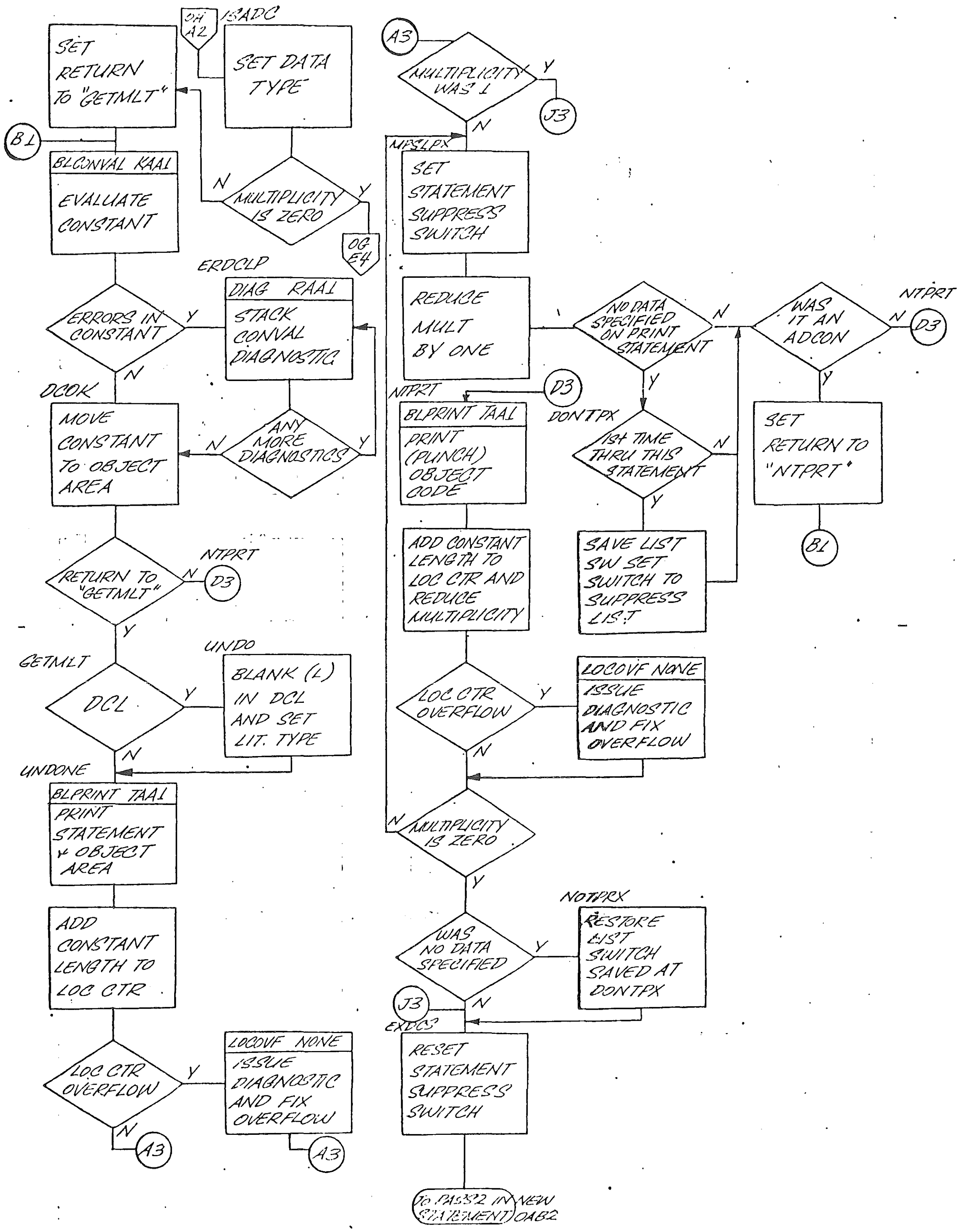
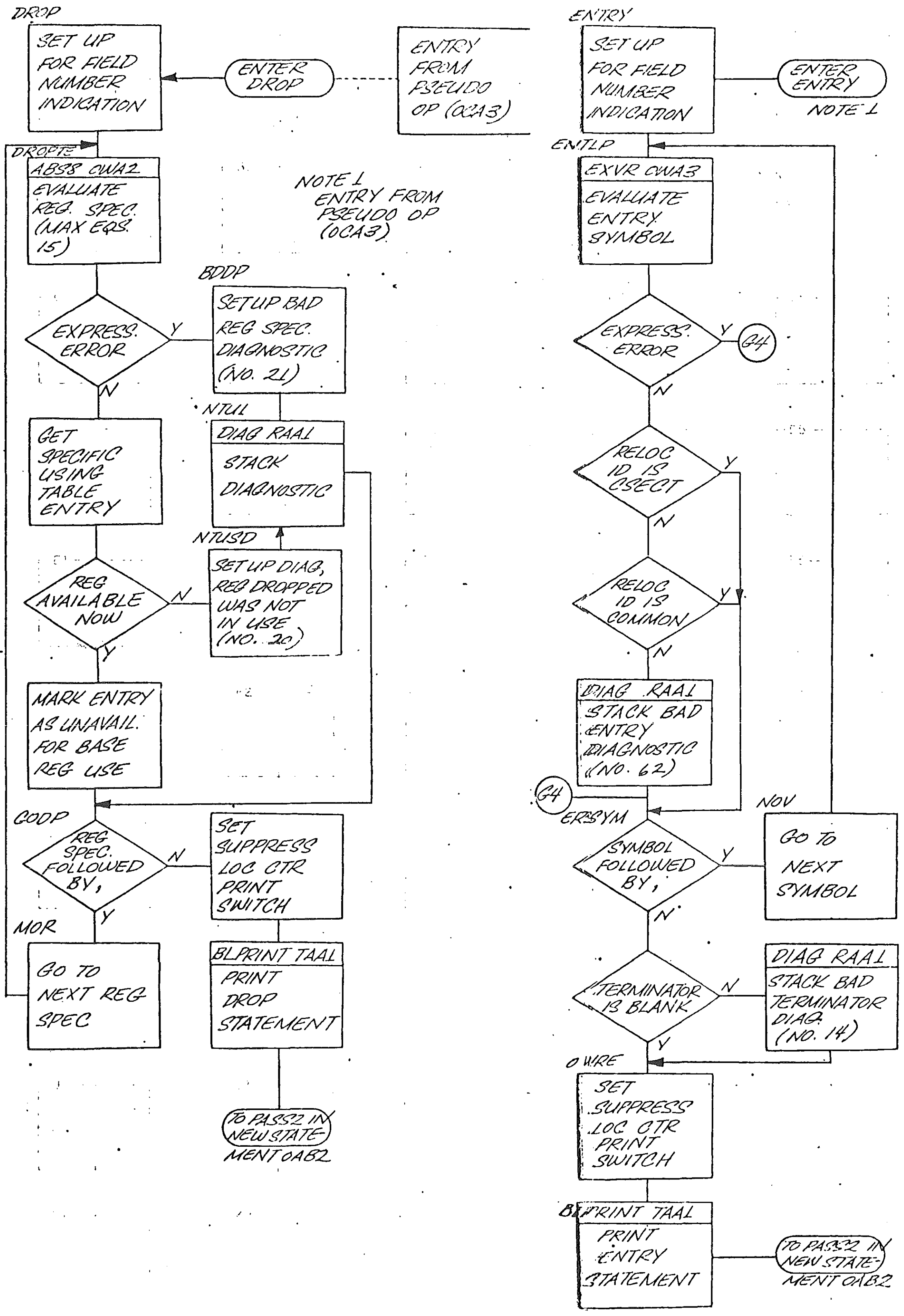

DCODS
SET DATA TYPE SET PARAMETERS FOR CONMOD
ENTER DCODS
NOTE 1
BLCONMOD GAAL
EVALUATE CONSTANT MODIFIERS
MODIFIER ERRORS
A3
MODIG
DIAG RAAL
STACK CONMOD DIAGNOSTIC
MORE DIAGNOSTICS
DCL
ILT
SET LIT. TYPE, BLANK (L) OF DCL
ERDCX
PREVENT OBJECT AREA PRINTING
BLPRINT TAAL
PRINT STATEMENT AND DIAGNOSTIC(S)
TO PASS 2 IN NEW STATEMENT OAB2
NOERM
TBL LOOKUP ALIGN SEQUENCE WITH CONMOD POINTER
WORD
DBLWD
BYTE
HFWD
SYLB
LOC CTR AT WORD
LOC CTR AT HFWRD
GET ALIGN. INCREMENT SET ALIGN. TYPE
HALF OXA3
ALIGN LOC CTR AT HALF-WORD BOUNDARY
BLPRINT TAAL
PRINT (PUNCH) ALIGN ZEROS
ADD INCREMENT TO LOC CTR
LOC CTR OVERFLOW
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
NOTE 1 ENTRY FROM PSEUDO OP (OCA3)
LOC CTR AT DBLWD
DOUB OXAL
ALIGN LOC CTR AT DOUBLE-WORD BOUNDARY
BYTES
VALUE LIST
DC OR DCL
OH A2
OG E4
ISADS
NOLDS
DS
SET DS TYPE
DIAG RAAL
STACK DC-NO VALUE LIST DIAG (NO. 19)
ISADSL
F4
ZERO OBJECT AREA COUNT TO FORCE PUNCH
BLPRINT TAAL
PRINT (PUNCH) STATEMENT
DCL
GETOL
BLANK (L) IN DCL AND SET LIT. TYPE
ADD LENGTH TO LOC CTR
LOC CTR OVERFLOW
LOCOVF
ISSUE DIAGNOSTIC AND FIX OVERFLOW
TO PASS 2 IN NEW STATEMENT OAB2
Y
N

CHART OG. DCODS SUBROUTINE (SHEET 1 OF 2)



SET RETURN TO "GETMLT"
OH A2
ISADC
SET DATA TYPE
MULTIPLICITY IS ZERO
N
Y
OG E4
B1
BLCONVAL KAA1
EVALUATE CONSTANT
ERRORS IN CONSTANT
Y
N
ERDCLP
DIAG RAA1
STACK CONVAL DIAGNOSTIC
ANY MORE DIAGNOSTICS
DCOK
MOVE CONSTANT TO OBJECT AREA
RETURN TO "GETMLT"
NTPRT
D3
GETMLT
DCL
UNDO
BLANK (L) IN DCL AND SET LIT. TYPE
UNDONE
BLPRINT TAA1
PRINT STATEMENT + OBJECT AREA
ADD CONSTANT LENGTH TO LOC CTR
LOC CTR OVERFLOW
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
A3
MULTIPLICITY WAS 1
J3
MFSLPX
SET STATEMENT SUPPRESS SWITCH
REDUCE MULT BY ONE
NTPRT
BLPRINT TAA1
PRINT (PUNCH) OBJECT CODE
ADD CONSTANT LENGTH TO LOC CTR AND REDUCE MULTIPLICITY
LOC CTR OVERFLOW
LOCOVF NONE
ISSUE DIAGNOSTIC AND FIX OVERFLOW
MULTIPLICITY IS ZERO
WAS NO DATA SPECIFIED
NOTPRX
RESTORE LIST SWITCH SAVED AT DONTPX
EXDCS
RESET STATEMENT SUPPRESS SWITCH
To PASS2 IN NEW STATEMENT OAB2
NO DATA SPECIFIED ON PRINT STATEMENT
DONTPX
1st TIME THRU THIS STATEMENT
SAVE LIST SW SET SWITCH TO SUPPRESS LIST
WAS IT AN ADCON
SET RETURN TO "NTPRT"

CHART OH. DCODS SUBROUTINE (SHEET 2 OF 2)



DROP
SET UP FOR FIELD NUMBER INDICATION
ENTER DROP
ENTRY FROM PSEUDO OP (0CA3)
DROPTE
ABSS CWA2
EVALUATE REG. SPEC. (MAX EQS. 15)
NOTE 1 ENTRY FROM PSEUDO OP (0CA3)
BDDP
EXPRESS. ERROR
Y
N
SET UP BAD REG SPEC. DIAGNOSTIC (NO. 21)
NTUL
DIAG RAAL
STACK DIAGNOSTIC
GET SPECIFIC USING TABLE ENTRY
NTUSD
REG AVAILABLE NOW
SET UP DIAG, REG DROPPED WAS NOT IN USE (NO. 20)
MARK ENTRY AS UNAVAIL. FOR BASE REG USE
COOP
REG SPEC. FOLLOWED BY ,
SET SUPPRESS LOC CTR PRINT SWITCH
MOR
GO TO NEXT REG SPEC
BLPRINT TAAL
PRINT DROP STATEMENT
TO PASS2 IN NEW STATEMENT 0AB2
ENTRY
SET UP FOR FIELD NUMBER INDICATION
ENTER ENTRY
NOTE 1
ENTLP
EXVR CWA3
EVALUATE ENTRY SYMBOL
EXPRESS. ERROR
G4
RELOC ID IS CSECT
RELOC ID IS COMMON
DIAG RAAL
STACK BAD ENTRY DIAGNOSTIC (NO. 62)
G4
ERSYM
NOV
SYMBOL FOLLOWED BY ,
GO TO NEXT SYMBOL
TERMINATOR IS BLANK
DIAG RAAL
STACK BAD TERMINATOR DIAG. (NO. 14)
OWRE
SET SUPPRESS LOC CTR PRINT SWITCH
BLPRINT TAAL
PRINT ENTRY STATEMENT
TO PASS2 IN NEW STATEMENT 0AB2

CHART OI. DROP, ENTRY SUBROUTINES

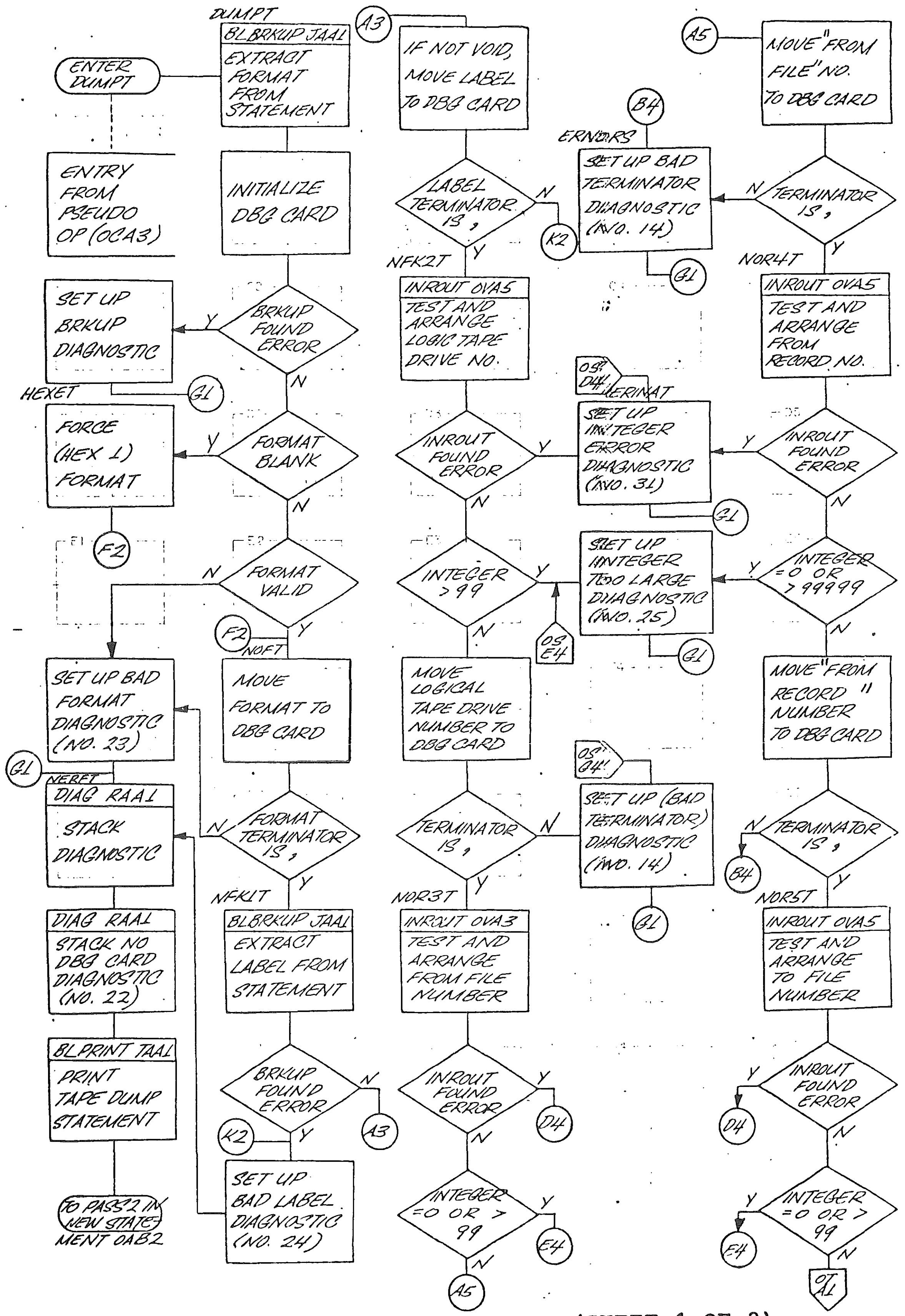

ENTER DUMPT
ENTRY FROM PSEUDO OP (0CA3)
DUMPT
BLBRKUP JAAL
EXTRACT FORMAT FROM STATEMENT
INITIALIZE DBG CARD
BRKUP FOUND ERROR
SET UP BRKUP DIAGNOSTIC
HEXET
FORCE (HEX 1) FORMAT
FORMAT BLANK
FORMAT VALID
NOFT
MOVE FORMAT TO DBG CARD
SET UP BAD FORMAT DIAGNOSTIC (NO. 23)
NERET
DIAG RAAL
STACK DIAGNOSTIC
STACK NO DBG CARD DIAGNOSTIC (NO. 22)
BLPRINT TAAL
PRINT TAPE DUMP STATEMENT
TO PASS 2 IN NEW STATEMENT 0AB2
FORMAT TERMINATOR IS ,
NFK1T
EXTRACT LABEL FROM STATEMENT
SET UP BAD LABEL DIAGNOSTIC (NO. 24)
IF NOT VOID, MOVE LABEL TO DBG CARD
LABEL TERMINATOR IS ,
NFK2T
INROUT OVA5
TEST AND ARRANGE LOGIC TAPE DRIVE NO.
INROUT FOUND ERROR
INTEGER > 99
MOVE LOGICAL TAPE DRIVE NUMBER TO DBG CARD
TERMINATOR IS ,
NOR3T
INROUT OVA3
TEST AND ARRANGE FROM FILE NUMBER
INTEGER =0 OR > 99
ERNORS
SET UP BAD TERMINATOR DIAGNOSTIC (NO. 14)
ERINAT
SET UP INTEGER ERROR DIAGNOSTIC (NO. 31)
SET UP INTEGER TOO LARGE DIAGNOSTIC (NO. 25)
SET UP (BAD TERMINATOR) DIAGNOSTIC (NO. 14)
MOVE "FROM FILE" NO. TO DBG CARD
TERMINATOR IS ,
NOR4T
TEST AND ARRANGE FROM RECORD NO.
INTEGER =0 OR > 99999
MOVE "FROM RECORD" NUMBER TO DBG CARD
NOR5T
TEST AND ARRANGE TO FILE NUMBER
OT A1
OS D4
OS E4

CHART OS. DUMPT SUBROUTINE (SHEET 1 OF 2)

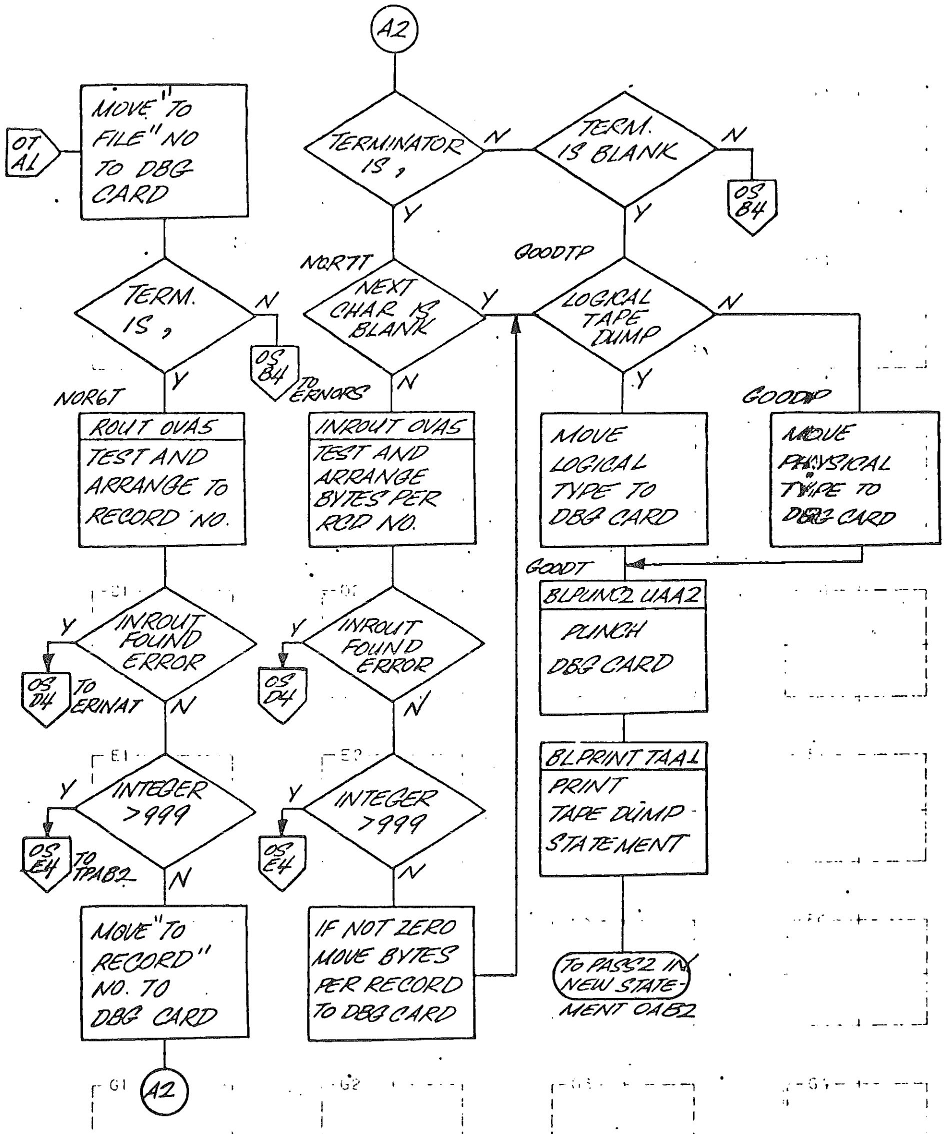


CHART OT. DUMPT SUBROUTINE (SHEET 2 OF 2)

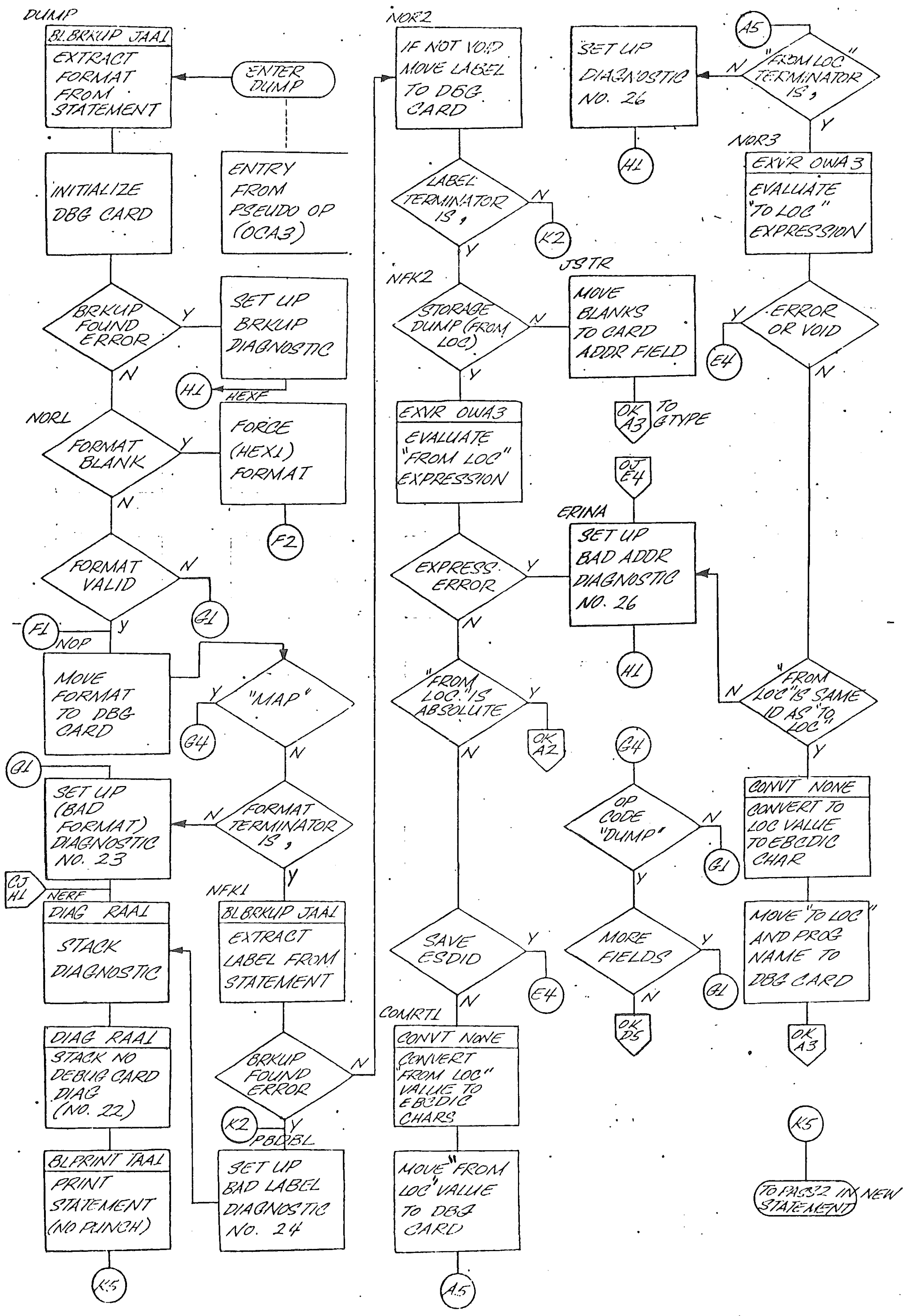
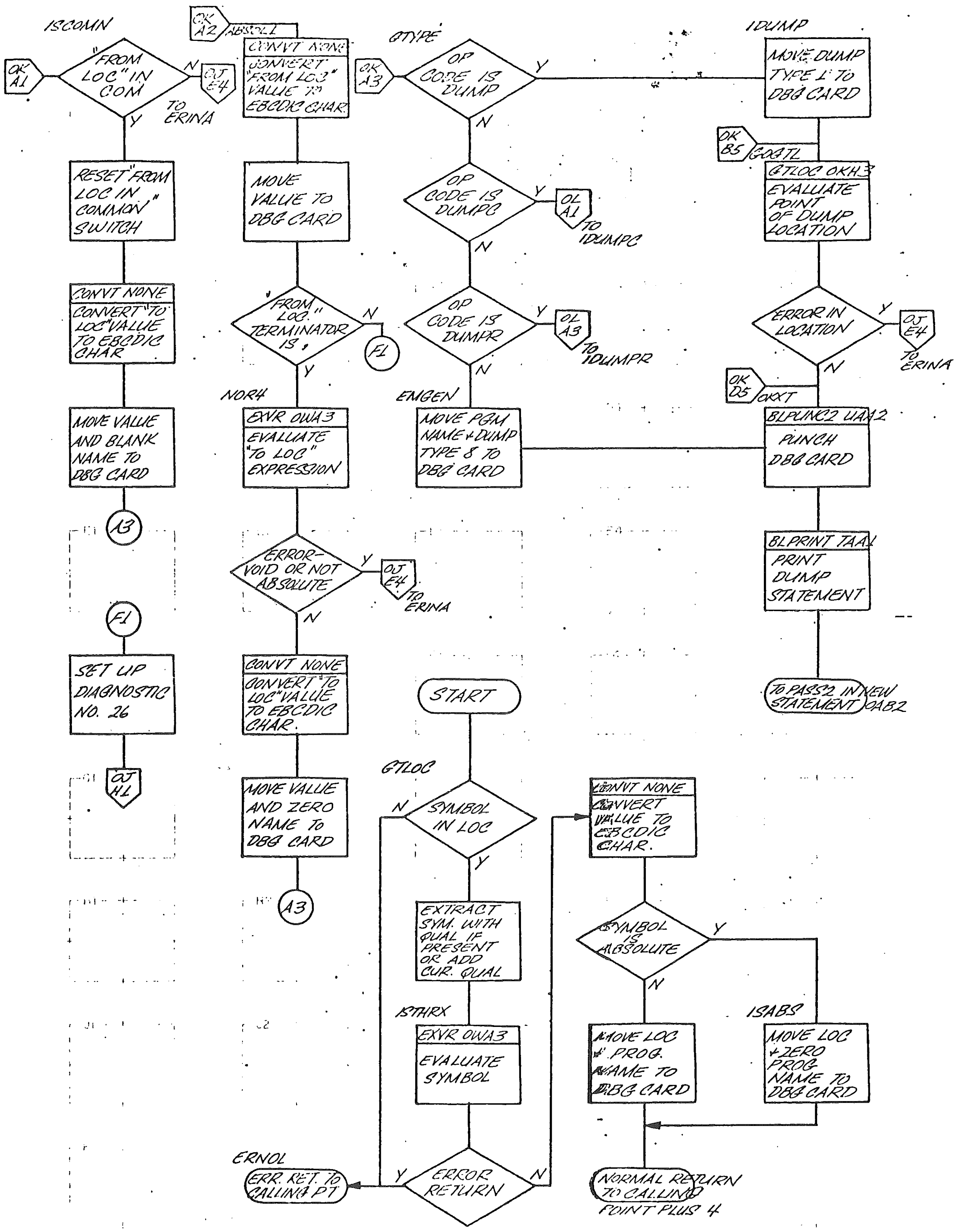

DUMP
BLBRKUP JAAL
EXTRACT FORMAT FROM STATEMENT
ENTER DUMP
ENTRY FROM PSEUDO OP (OCA3)
INITIALIZE DBG CARD
BRKUP FOUND ERROR
SET UP BRKUP DIAGNOSTIC
H1
HEXF
NOR1
FORMAT BLANK
FORCE (HEX1) FORMAT
F2
FORMAT VALID
G1
F1
NOP
MOVE FORMAT TO DBG CARD
"MAP"
G4
SET UP (BAD FORMAT) DIAGNOSTIC NO. 23
FORMAT TERMINATOR IS ,
CJ H1
NERF
DIAG RAA1
STACK DIAGNOSTIC
NFK1
BLBRKUP JAA1
EXTRACT LABEL FROM STATEMENT
DIAG RAA1
STACK NO DEBUG CARD DIAG (NO. 22)
BRKUP FOUND ERROR
K2
PBDBL
BLPRINT TAA1
PRINT STATEMENT (NO PUNCH)
SET UP BAD LABEL DIAGNOSTIC NO. 24
K5
NOR2
IF NOT VOID MOVE LABEL TO DBG CARD
LABEL TERMINATOR IS ,
K2
NFK2
STORAGE DUMP (FROM LOC)
JSTR
MOVE BLANKS TO CARD ADDR FIELD
OK A3
TO GTYPE
EXVR OWA3
EVALUATE "FROM LOC" EXPRESSION
OJ E4
ERINA
EXPRESS ERROR
SET UP BAD ADDR DIAGNOSTIC NO. 26
H1
"FROM LOC" IS ABSOLUTE
OK A2
G4
OP CODE "DUMP"
G1
SAVE ESDID
E4
MORE FIELDS
G1
OK D5
COMRT1
CONVT NONE
CONVERT "FROM LOC" VALUE TO EBCDIC CHARS
MOVE "FROM LOC" VALUE TO DBG CARD
A5
SET UP DIAGNOSTIC NO. 26
A5
"FROM LOC" TERMINATOR IS ,
H1
NOR3
EXVR OWA3
EVALUATE "TO LOC" EXPRESSION
ERROR OR VOID
E4
"FROM LOC" IS SAME ID AS "TO LOC"
CONVT NONE
CONVERT TO LOC VALUE TO EBCDIC CHAR
MOVE "TO LOC" AND PROG NAME TO DBG CARD
OK A3
K5
TO PASS2 IN NEW STATEMENT
Y
N

CHART OJ. DUMP SUBROUTINE (SHEET 1 OF 3)



ISCOMN
OK A1
"FROM LOC" IN COM
N
OJ E4 TO ERINA
Y
RESET "FROM LOC IN COMMON" SWITCH
CONVT NONE
CONVERT "TO LOC" VALUE TO EBCDIC CHAR
MOVE VALUE AND BLANK NAME TO DBG CARD
A3
F1
SET UP DIAGNOSTIC NO. 26
OJ H1
OK A2
ABSOLL
CONVT NONE
CONVERT "FROM LOC" VALUE TO EBCDIC CHAR
MOVE VALUE TO DBG CARD
"FROM LOC." TERMINATOR IS ,
N
F1
Y
NOR4
EXVR OWA3
EVALUATE "TO LOC" EXPRESSION
ERROR- VOID OR NOT ABSOLUTE
Y
OJ E4 TO ERINA
N
CONVT NONE
CONVERT "TO LOC" VALUE TO EBCDIC CHAR.
MOVE VALUE AND ZERO NAME TO DBG CARD
A3
OTYPE
OK A3
OP CODE IS DUMP
Y
N
OP CODE IS DUMPC
Y
OL A1 TO IDUMPC
N
OP CODE IS DUMPR
Y
OL A3 TO IDUMPR
N
EMGEN
MOVE PGM NAME + DUMP TYPE 8 TO DBG CARD
IDUMP
MOVE DUMP TYPE 1 TO DBG CARD
OK B5
GOGTL
GTLOC OKH3
EVALUATE POINT OF DUMP LOCATION
ERROR IN LOCATION
Y
OJ E4 TO ERINA
N
OK D5
OKXT
BLPUNC2 UAA2
PUNCH DBG CARD
BLPRINT TAA1
PRINT DUMP STATEMENT
TO PASS2 IN NEW STATEMENT OAB2
START
GTLOC
SYMBOL IN LOC
N
Y
EXTRACT SYM. WITH QUAL IF PRESENT OR ADD CUR. QUAL
ISTHRX
EXVR OWA3
EVALUATE SYMBOL
ERROR RETURN
Y
N
ERNOL
ERR. RET. TO CALLING PT
CONVT NONE
CONVERT VALUE TO EBCDIC CHAR.
SYMBOL IS ABSOLUTE
Y
N
MOVE LOC + PROG. NAME TO DBG CARD
ISABS
MOVE LOC + ZERO PROG. NAME TO DBG CARD
NORMAL RETURN TO CALLING POINT PLUS 4

CHART OK. DUMP SUBROUTINE (SHEET 2 OF 3)

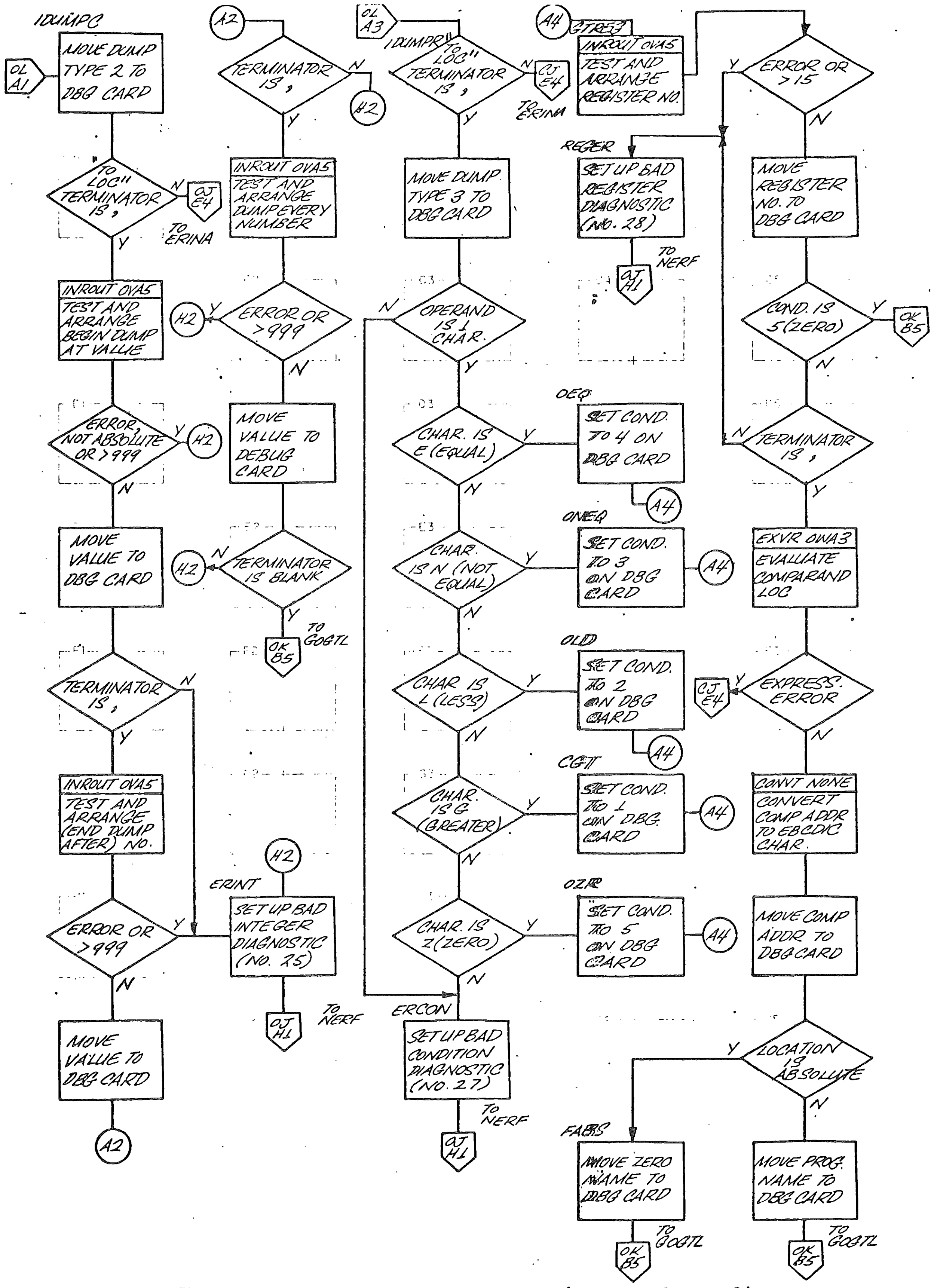
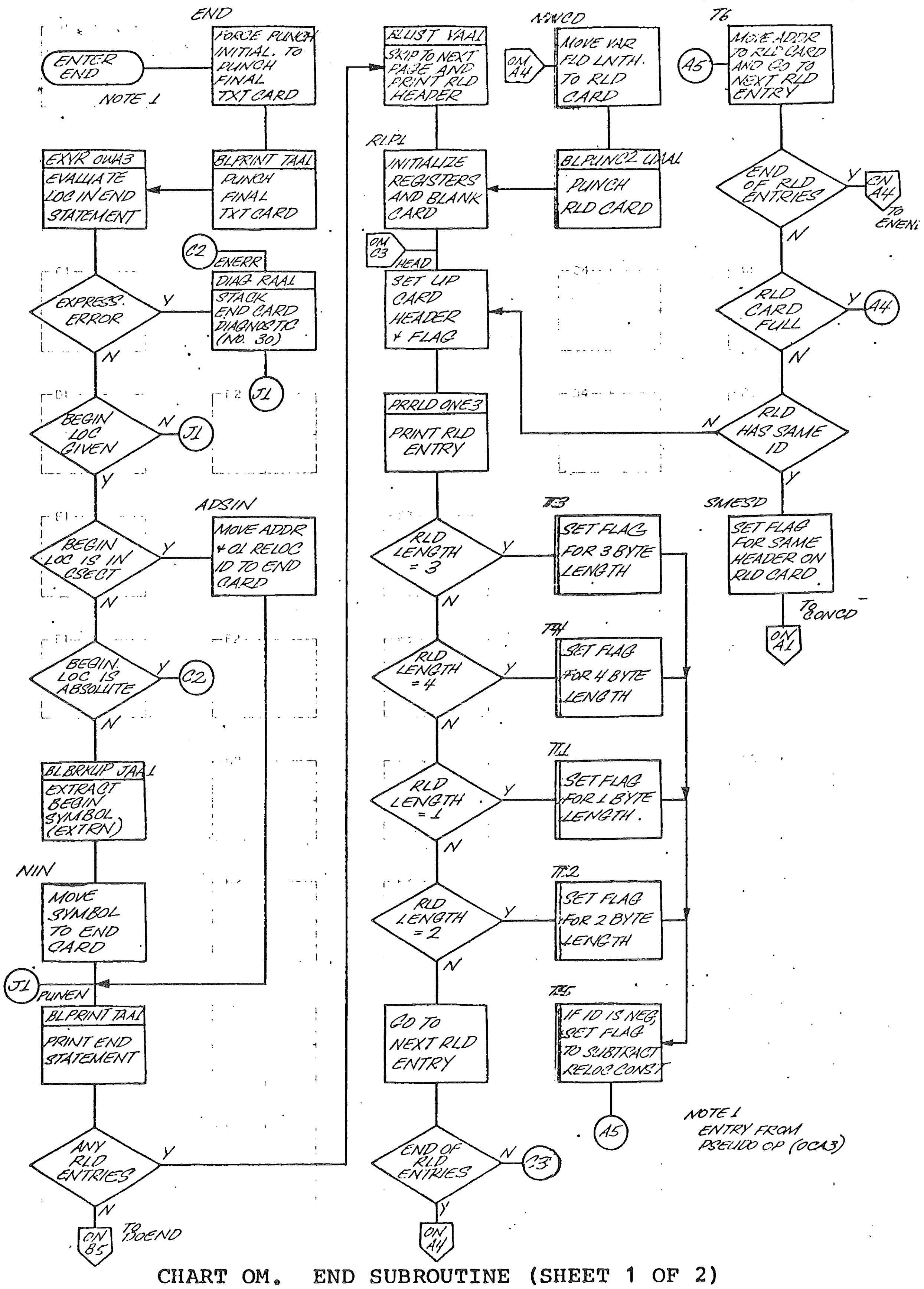
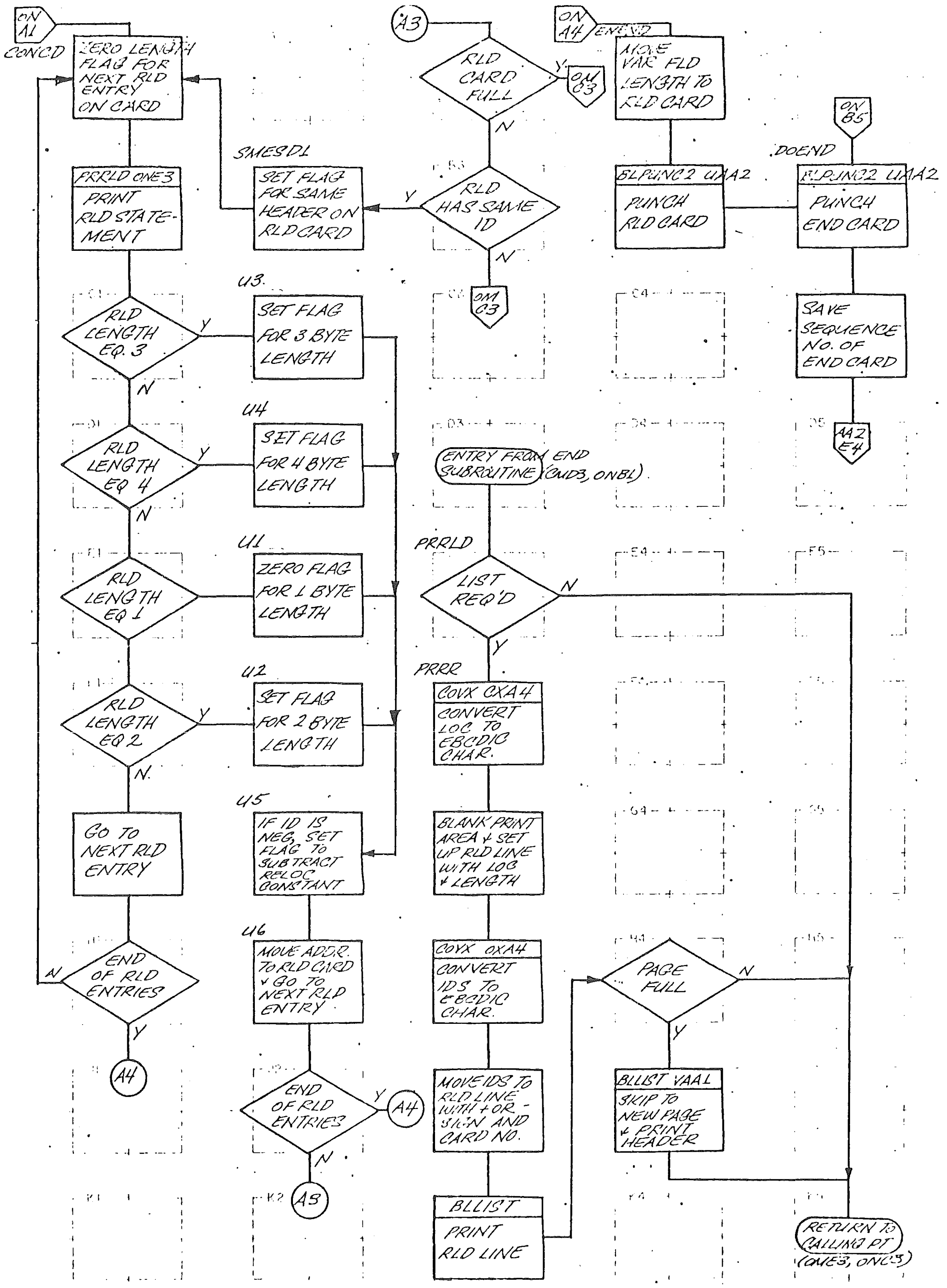

IDUMPC
OL A1
MOVE DUMP TYPE 2 TO DBG CARD
TO "LOC" TERMINATOR IS ,
N
CJ E4
TO ERINA
Y
INROUT OVA5 TEST AND ARRANGE BEGIN DUMP AT VALUE
ERROR, NOT ABSOLUTE OR >999
Y
H2
N
MOVE VALUE TO DBG CARD
TERMINATOR IS ;
N
Y
INROUT OVA5 TEST AND ARRANGE (END DUMP AFTER) NO.
ERROR OR >999
Y
N
MOVE VALUE TO DBG CARD
A2
A2
TERMINATOR IS ,
N
H2
Y
INROUT OVA5 TEST AND ARRANGE DUMP EVERY NUMBER
ERROR OR >999
Y
H2
N
MOVE VALUE TO DEBUG CARD
TERMINATOR IS BLANK
N
H2
Y
OK B5
TO GOGTL
H2
ERINT
SET UP BAD INTEGER DIAGNOSTIC (NO. 25)
OJ H1
TO NERF
OL A3
IDUMPR
TO "LOC" TERMINATOR IS ,
N
CJ E4
TO ERINA
Y
MOVE DUMP TYPE 3 TO DBG CARD
OPERAND IS 1 CHAR.
N
Y
CHAR. IS E (EQUAL)
Y
N
CHAR. IS N (NOT EQUAL)
Y
N
CHAR IS L (LESS)
Y
N
CHAR. IS G (GREATER)
Y
N
CHAR. IS Z (ZERO)
Y
N
ERCON
SET UP BAD CONDITION DIAGNOSTIC (NO. 27)
OJ H1
TO NERF
A4
STRES
INROUT OVA5 TEST AND ARRANGE REGISTER NO.
REGER
SET UP BAD REGISTER DIAGNOSTIC (NO. 28)
OJ H1
TO NERF
OEQ
SET COND. TO 4 ON DBG CARD
A4
ONEQ
SET COND. TO 3 ON DBG CARD
A4
OLD
SET COND. TO 2 ON DBG CARD
A4
OGT
SET COND. TO 1 ON DBG CARD
A4
OZR
SET COND. TO 5 ON DBG CARD
A4
ERROR OR >15
Y
N
MOVE REGISTER NO. TO DBG CARD
COND. IS 5 (ZERO)
Y
OK B5
N
TERMINATOR IS ,
N
Y
EXVR OWA3 EVALUATE COMPARAND LOC
EXPRESS. ERROR
Y
CJ E4
N
CONVT NONE CONVERT COMP ADDR TO EBCDIC CHAR.
MOVE COMP ADDR TO DBG CARD
LOCATION IS ABSOLUTE
Y
N
FABS
MOVE ZERO NAME TO DBG CARD
OK B5
TO GOGTL
MOVE PROG. NAME TO DBG CARD
OK B5
TO GOGTL

CHART OL. DUMP SUBROUTINE (SHEET 3 OF 3)



ENTER END
NOTE 1
END
FORCE PUNCH INITIAL. TO PUNCH FINAL TXT CARD
BLPRINT TAAL
PUNCH FINAL TXT CARD
EXVR OWA3
EVALUATE LOC IN END STATEMENT
EXPRESS. ERROR
C2
ENERR
DIAG RAAL
STACK END CARD DIAGNOSTIC (NO. 30)
J1
BEGIN LOC GIVEN
ADSIN
MOVE ADDR + 01 RELOC ID TO END CARD
BEGIN LOC IS IN CSECT
BEGIN. LOC IS ABSOLUTE
BLBRKUP JAAL
EXTRACT BEGIN SYMBOL (EXTRN)
NIN
MOVE SYMBOL TO END CARD
PUNEN
BLPRINT TAAL
PRINT END STATEMENT
ANY RLD ENTRIES
ON B5
TO DOEND
BLLIST VAAL
SKIP TO NEXT PAGE AND PRINT RLD HEADER
RLPL
INITIALIZE REGISTERS AND BLANK CARD
ON C3
HEAD
SET UP CARD HEADER + FLAG
PRRLD ONE3
PRINT RLD ENTRY
RLD LENGTH = 3
RLD LENGTH = 4
RLD LENGTH = 1
RLD LENGTH = 2
GO TO NEXT RLD ENTRY
END OF RLD ENTRIES
C3
ON A4
NSWCD
ON A4
MOVE VAR FLD LNTH. TO RLD CARD
BLPUNC2 UAAL
PUNCH RLD CARD
T3
SET FLAG FOR 3 BYTE LENGTH
T4
SET FLAG FOR 4 BYTE LENGTH
T1
SET FLAG FOR 1 BYTE LENGTH.
T2
SET FLAG FOR 2 BYTE LENGTH
T5
IF ID IS NEG, SET FLAG TO SUBTRACT RELOC CONST
A5
T6
A5
MOVE ADDR TO RLD CARD AND GO TO NEXT RLD ENTRY
END OF RLD ENTRIES
ON A4
TO ENEN
RLD CARD FULL
A4
RLD HAS SAME ID
SMESD
SET FLAG FOR SAME HEADER ON RLD CARD
TO CONCD
ON A1
NOTE 1
ENTRY FROM PSEUDO OP (OCA3)
Y
N

CHART OM. END SUBROUTINE (SHEET 1 OF 2)



ON A1
CONCD
ZERO LENGTH FLAG FOR NEXT RLD ENTRY ON CARD
PRRLD ONE3
PRINT RLD STATEMENT
RLD LENGTH EQ. 3
RLD LENGTH EQ 4
RLD LENGTH EQ 1
RLD LENGTH EQ 2
GO TO NEXT RLD ENTRY
END OF RLD ENTRIES
A4
SMESD1
SET FLAG FOR SAME HEADER ON RLD CARD
U3
SET FLAG FOR 3 BYTE LENGTH
U4
SET FLAG FOR 4 BYTE LENGTH
U1
ZERO FLAG FOR 1 BYTE LENGTH
U2
SET FLAG FOR 2 BYTE LENGTH
U5
IF ID IS NEG, SET FLAG TO SUBTRACT RELOC CONSTANT
U6
MOVE ADDR. TO RLD CARD & GO TO NEXT RLD ENTRY
END OF RLD ENTRIES
A4
A3
A3
RLD CARD FULL
OM 03
RLD HAS SAME ID
OM 03
ENTRY FROM END SUBROUTINE (OND3, ONB1)
PRRLD
LIST REQ'D
PRRR
COVX CXA4
CONVERT LOC TO EBCDIC CHAR.
BLANK PRINT AREA & SET UP RLD LINE WITH LOC & LENGTH
COVX CXA4
CONVERT IDS TO EBCDIC CHAR.
MOVE IDS TO RLD LINE WITH + OR - SIGN AND CARD NO.
BLLIST
PRINT RLD LINE
ON A4
ENEND
MOVE VAR FLD LENGTH TO RLD CARD
BLPUNC2 UAA2
PUNCH RLD CARD
PAGE FULL
BLLIST VAA1
SKIP TO NEW PAGE & PRINT HEADER
ON B5
.DOEND
BLPUNC2 UAA2
PUNCH END CARD
SAVE SEQUENCE NO. OF END CARD
AA2 E4
RETURN TO CALLING PT (ONE3, ONC3)
Y
N

CHART ON. END SUBROUTINE (SHEET 2 OF 2)

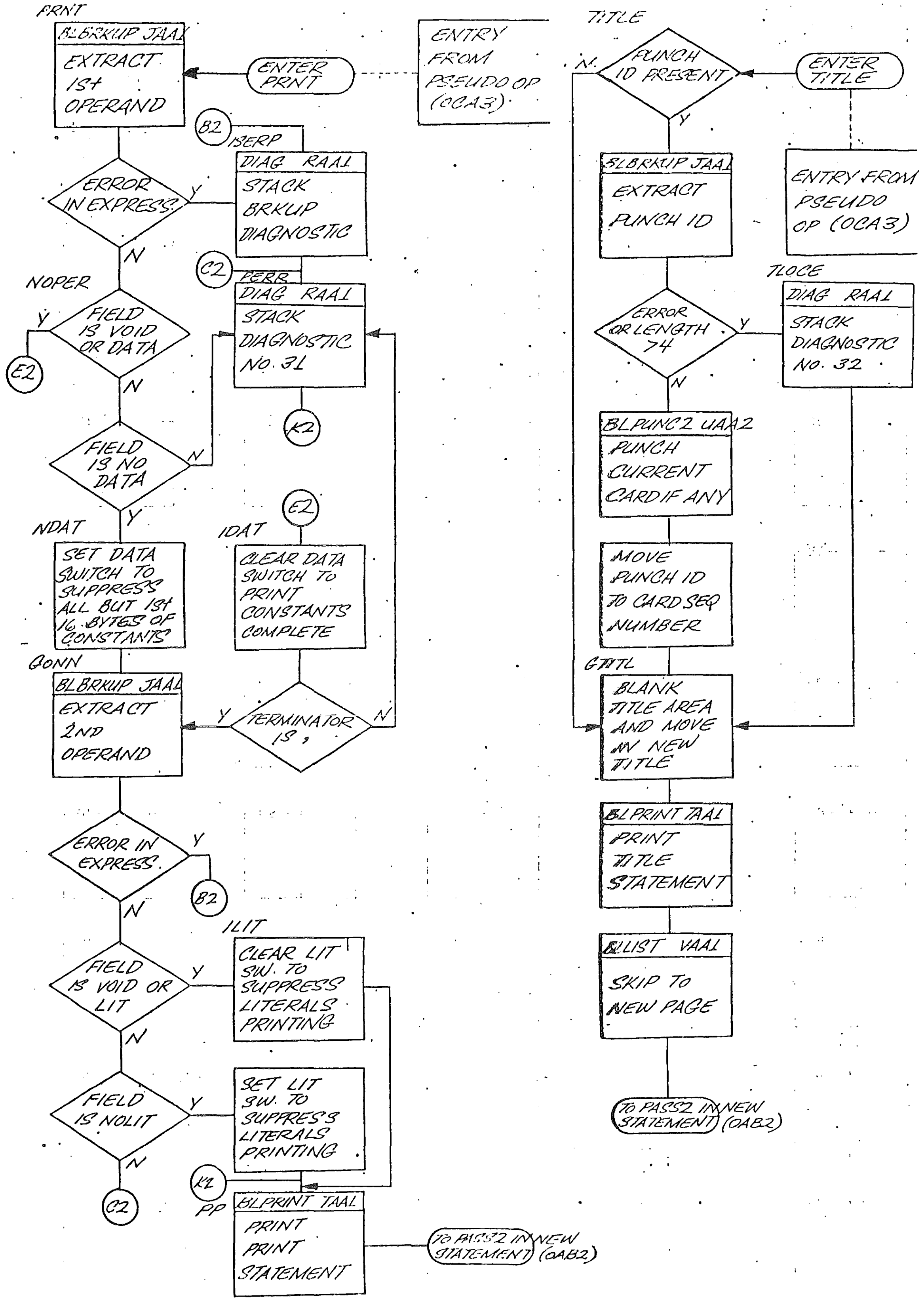


CHART OO. PRNT, TITLE SUBROUTINES

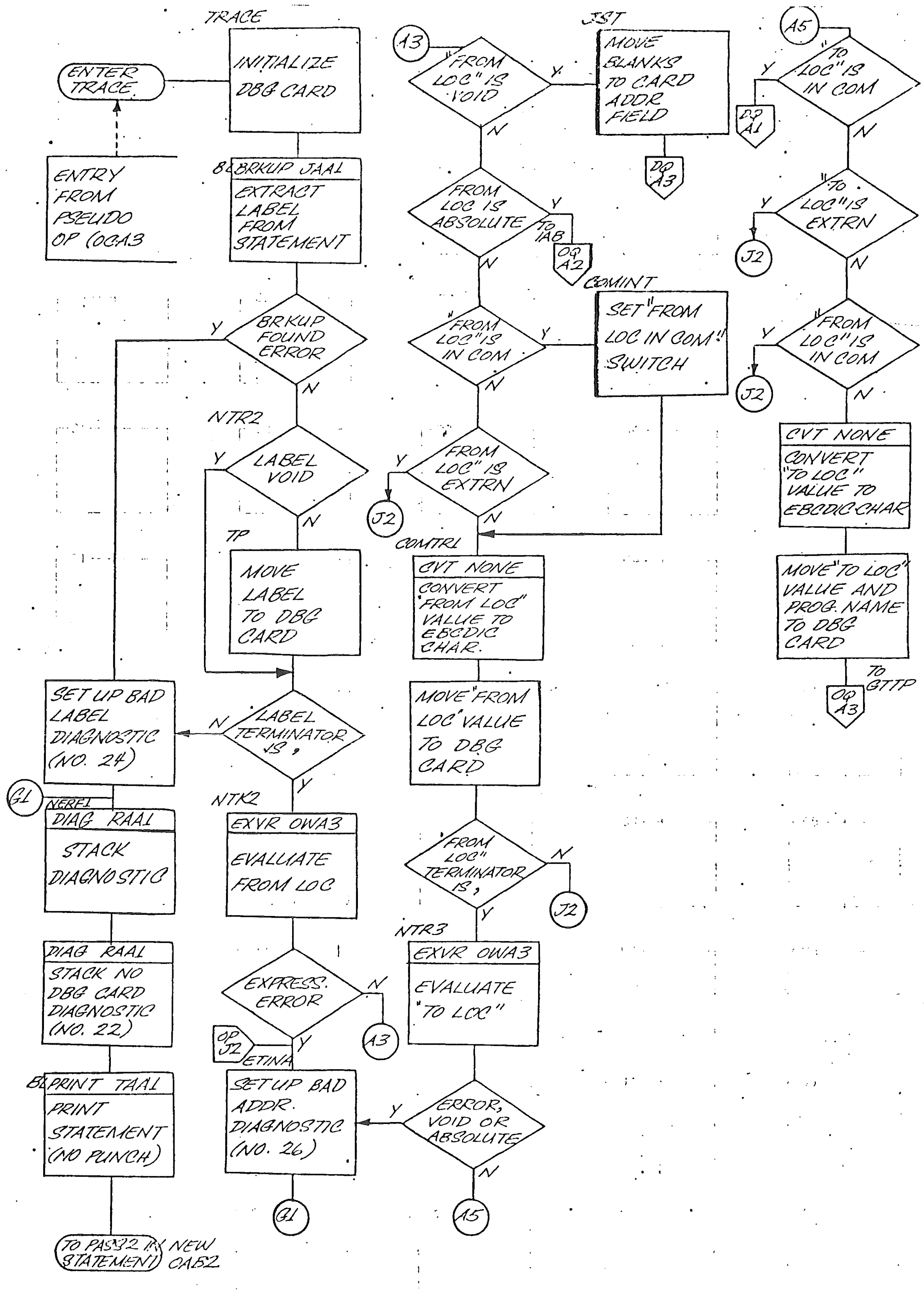
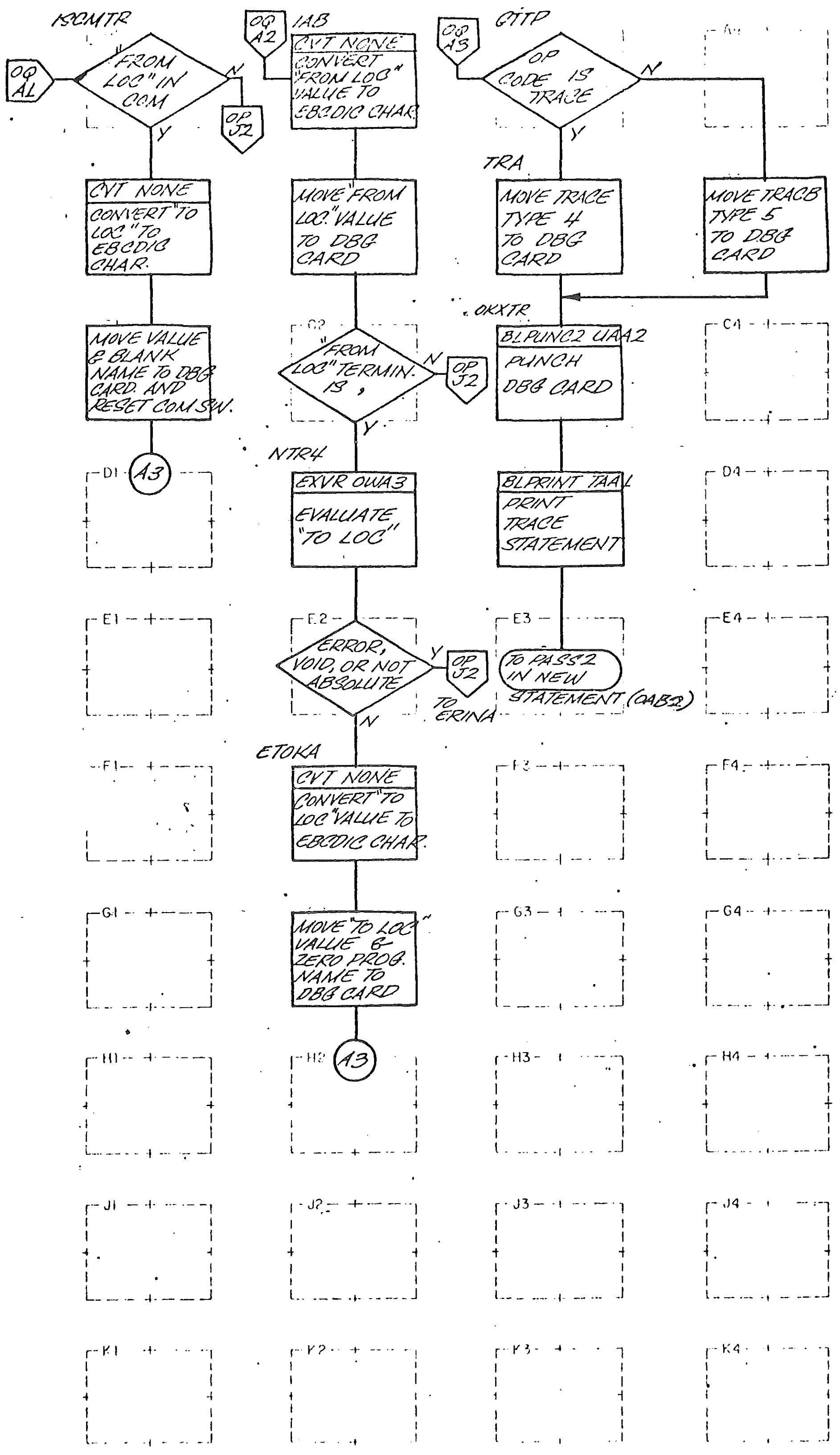


CHART OP. TRACE SUBROUTINE (SHEET 1 OF 2)



ISCOMTR
OQ AL
FROM LOC "IN" COM
N
OP J2
Y
CVT NONE
CONVERT "TO LOC" TO EBCDIC CHAR.
MOVE VALUE & BLANK NAME TO DBG CARD AND RESET COM SW.
A3
OQ A2
IAB
CVT NONE
CONVERT "FROM LOC" VALUE TO EBCDIC CHAR
MOVE "FROM LOC" VALUE TO DBG CARD
"FROM LOC" TERMIN. IS ,
N
OP J2
Y
NTR4
EXVR OWA3
EVALUATE "TO LOC"
ERROR, VOID, OR NOT ABSOLUTE
Y
OP J2
TO ERINA
N
ETOKA
CVT NONE
CONVERT "TO LOC" VALUE TO EBCDIC CHAR.
MOVE "TO LOC" VALUE & ZERO PROG. NAME TO DBG CARD
A3
OQ A3
OTTP
OP CODE IS TRACE
N
Y
TRA
MOVE TRACE TYPE 4 TO DBG CARD
MOVE TRACB TYPE 5 TO DBG CARD
OKXTR
BLPUNC2 UAA2
PUNCH DBG CARD
BLPRINT TAA1
PRINT TRACE STATEMENT
TO PASS 2 IN NEW STATEMENT (OAB2)

CHART OQ. TRACE SUBROUTINE (SHEET 2 OF 2)

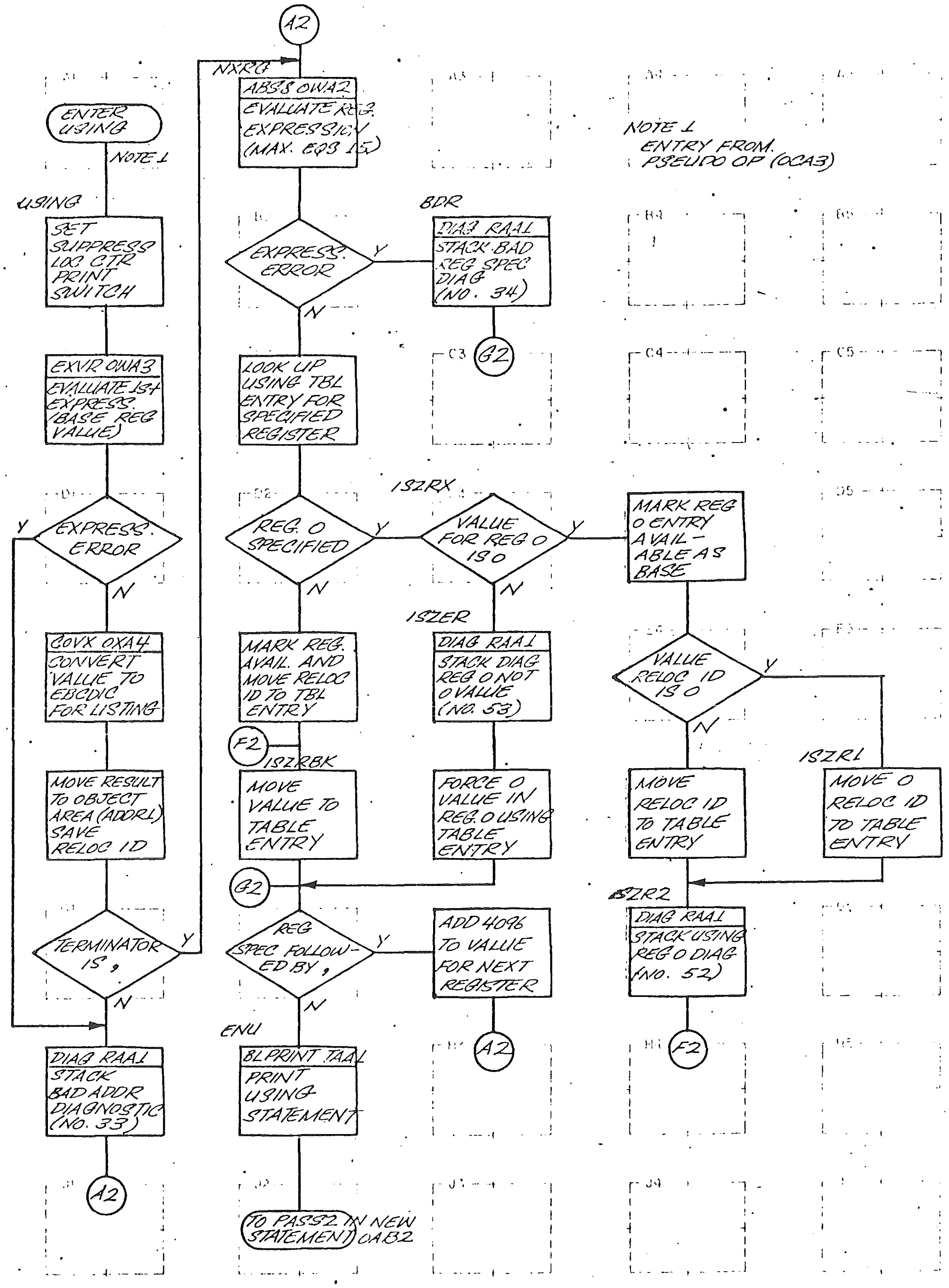

ENTER USING
NOTE 1
USING
SET SUPPRESS LOC CTR PRINT SWITCH
EXVR OWA3
EVALUATE 1st EXPRESS. (BASE REG VALUE)
EXPRESS. ERROR
COVX OXA4
CONVERT VALUE TO EBCDIC FOR LISTING
MOVE RESULT TO OBJECT AREA (ADDR1) SAVE RELOC ID
TERMINATOR IS ,
DIAG RAA1
STACK BAD ADDR DIAGNOSTIC (NO. 33)
A2
NXRG
ABSS OWA2
EVALUATE REG. EXPRESSION (MAX. EQS 15)
EXPRESS. ERROR
BDR
DIAG RAA1
STACK BAD REG SPEC DIAG (NO. 34)
G2
LOOK UP USING TBL ENTRY FOR SPECIFIED REGISTER
REG. 0 SPECIFIED
ISZRX
VALUE FOR REG 0 IS 0
MARK REG 0 ENTRY AVAIL-ABLE AS BASE
MARK REG. AVAIL. AND MOVE RELOC ID TO TBL ENTRY
ISZER
DIAG RAA1
STACK DIAG REG 0 NOT 0 VALUE (NO. 53)
VALUE RELOC ID IS 0
F2
ISZRBK
MOVE VALUE TO TABLE ENTRY
FORCE 0 VALUE IN REG. 0 USING TABLE ENTRY
MOVE RELOC ID TO TABLE ENTRY
ISZR1
MOVE 0 RELOC ID TO TABLE ENTRY
ISZR2
DIAG RAA1
STACK USING REG 0 DIAG (NO. 52)
REG SPEC FOLLOW-ED BY ,
ADD 4096 TO VALUE FOR NEXT REGISTER
ENU
BLPRINT TAA1
PRINT USING STATEMENT
TO PASS2 IN NEW STATEMENT OAB2
NOTE 1
ENTRY FROM PSEUDO OP (OCA3)
Y
N

CHART OR. USING SUBROUTINE

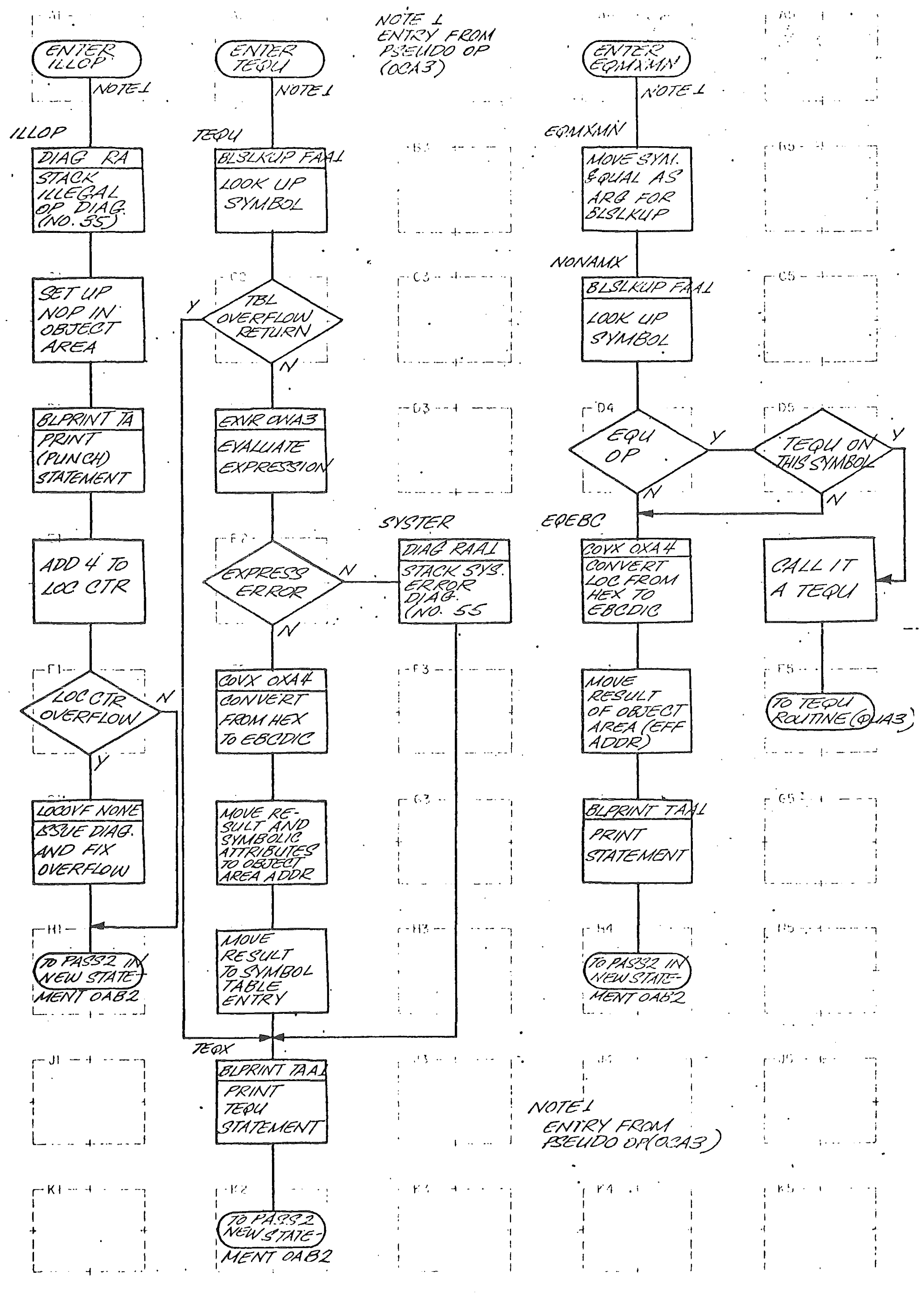

ENTER ILLOP
NOTE 1
ILLOP
DIAG RA
STACK ILLEGAL OP DIAG. (NO. 55)
SET UP NOP IN OBJECT AREA
BLPRINT TA
PRINT (PUNCH) STATEMENT
ADD 4 TO LOC CTR
LOC CTR OVERFLOW
N
Y
LOCOVF NONE
ISSUE DIAG. AND FIX OVERFLOW
TO PASS2 IN NEW STATEMENT OAB2
ENTER TEQU
NOTE 1
TEQU
BLSLKUP FAAL
LOOK UP SYMBOL
TBL OVERFLOW RETURN
Y
N
EXVR OWA3
EVALUATE EXPRESSION
EXPRESS ERROR
N
N
SYSTER
DIAG RAAL
STACK SYS. ERROR DIAG. (NO. 55)
COVX OXA4
CONVERT FROM HEX TO EBCDIC
MOVE RESULT AND SYMBOLIC ATTRIBUTES TO OBJECT AREA ADDR
MOVE RESULT TO SYMBOL TABLE ENTRY
TEQX
BLPRINT TAAL
PRINT TEQU STATEMENT
TO PASS2 NEW STATEMENT OAB2
NOTE 1 ENTRY FROM PSEUDO OP (OCA3)
ENTER EQMXMN
NOTE 1
EQMXMN
MOVE SYM. EQUAL AS ARG FOR BLSLKUP
NONAMX
BLSLKUP FAAL
LOOK UP SYMBOL
EQU OP
Y
N
TEQU ON THIS SYMBOL
Y
N
EQEBC
COVX OXA4
CONVERT LOC FROM HEX TO EBCDIC
CALL IT A TEQU
TO TEQU ROUTINE (OUA3)
MOVE RESULT OF OBJECT AREA (EFF ADDR)
BLPRINT TAAL
PRINT STATEMENT
TO PASS2 IN NEW STATEMENT OAB2
NOTE 1 ENTRY FROM PSEUDO OP (OCA3)

CHART OU. ILLOP, TEQU, EQMXMN SUBROUTINES

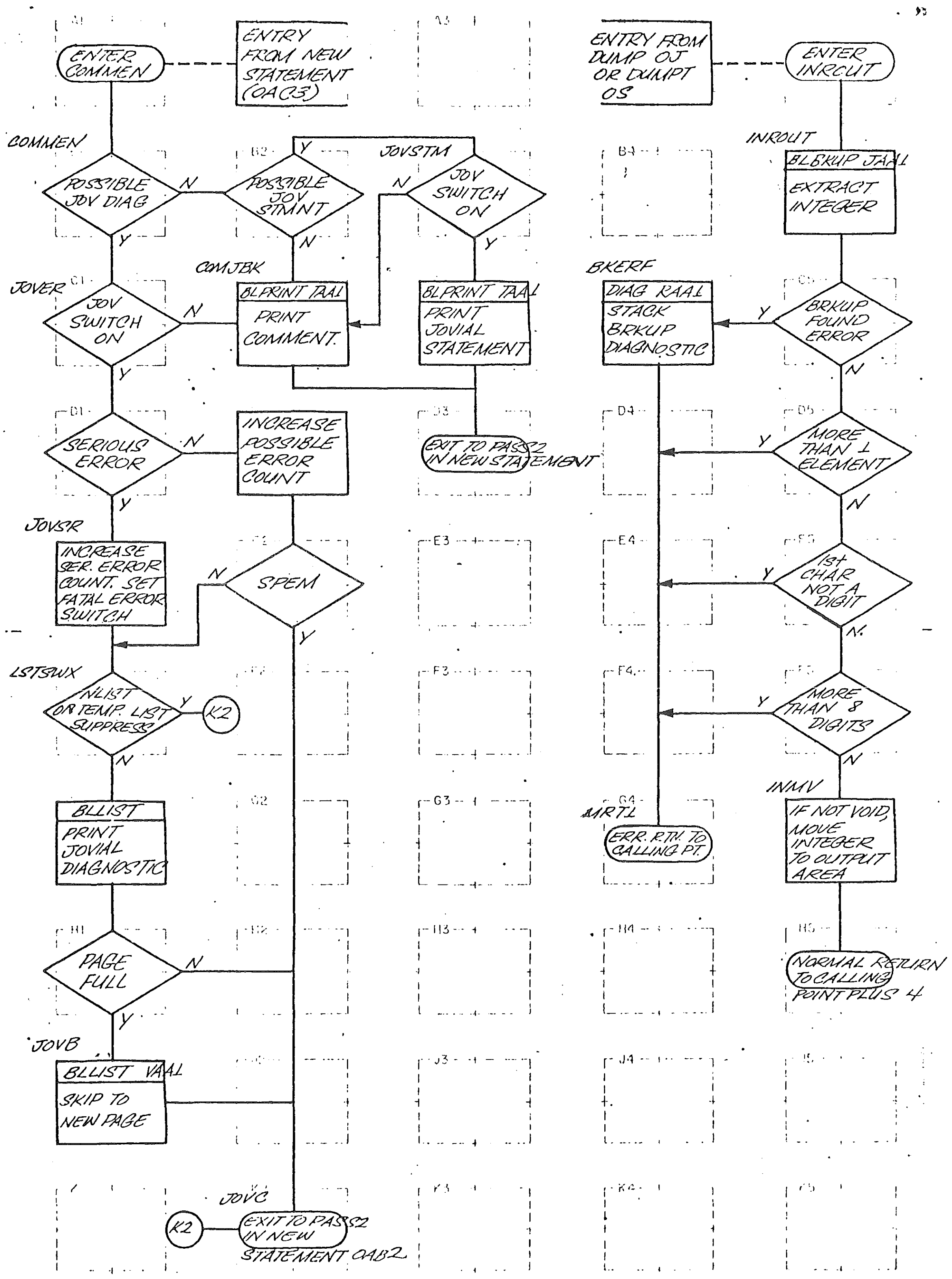


CHART OV. COMMEN, INROUT SUBROUTINES

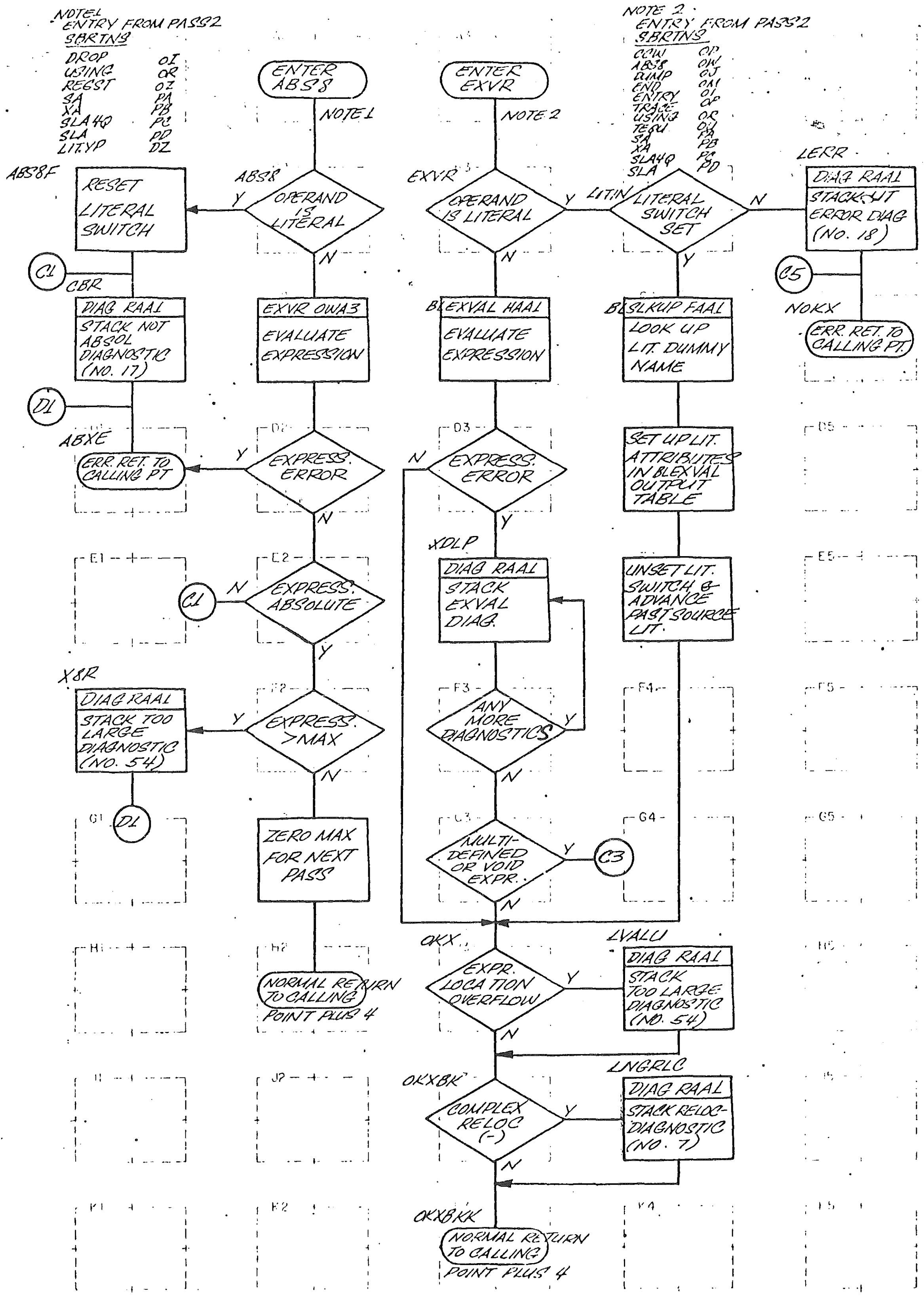


CHART OW. ABS8, EXVR, LOCOVF SUBROUTINES

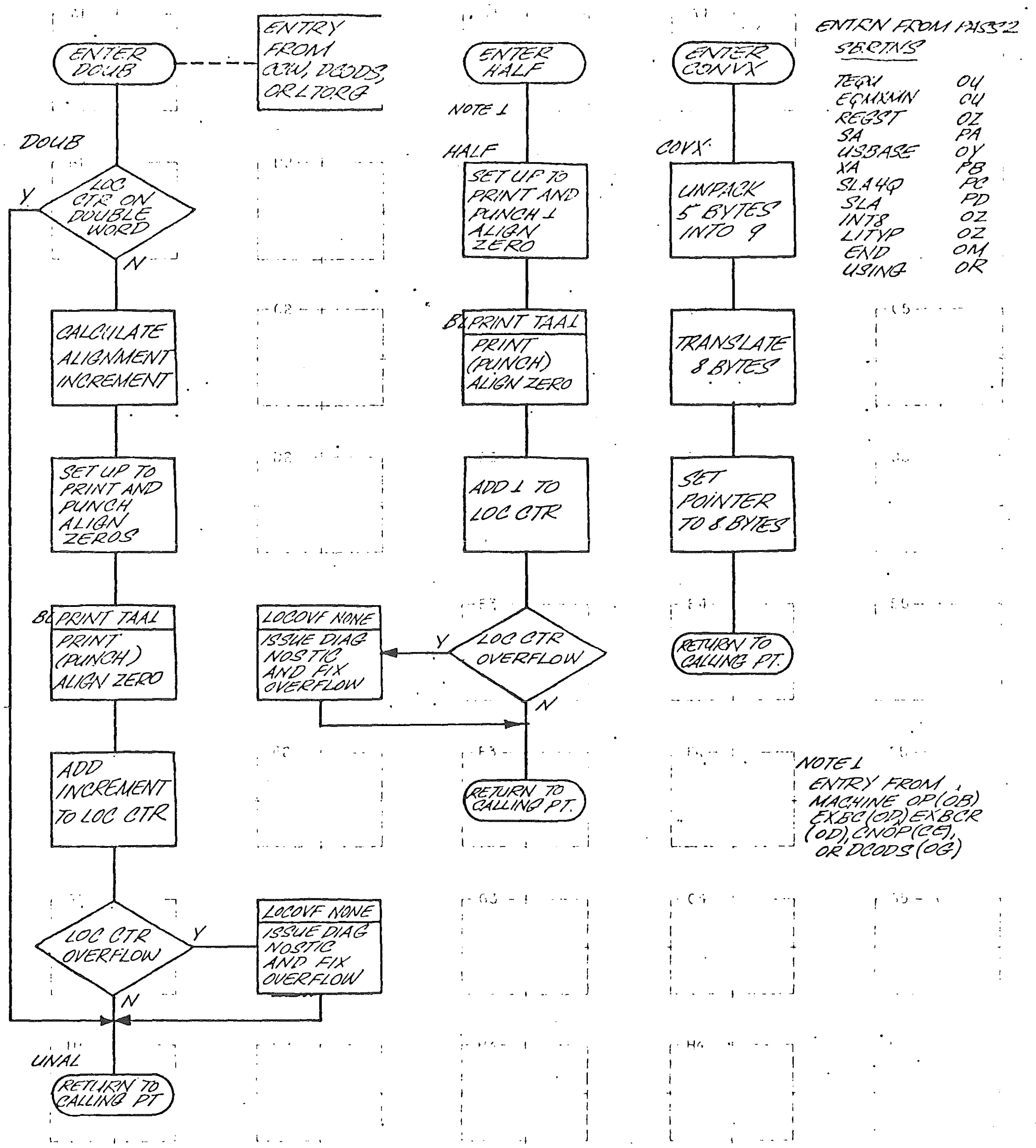
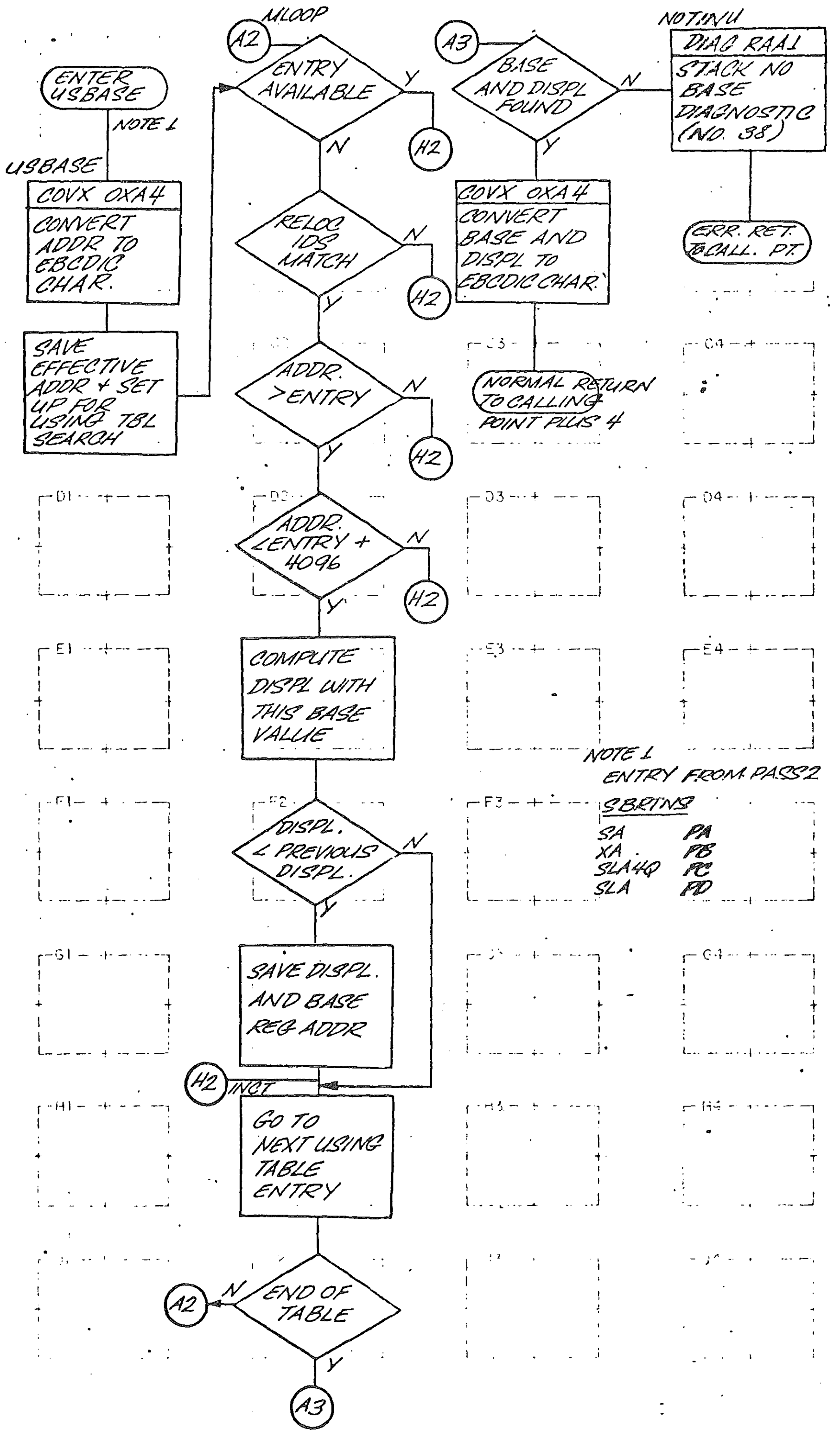
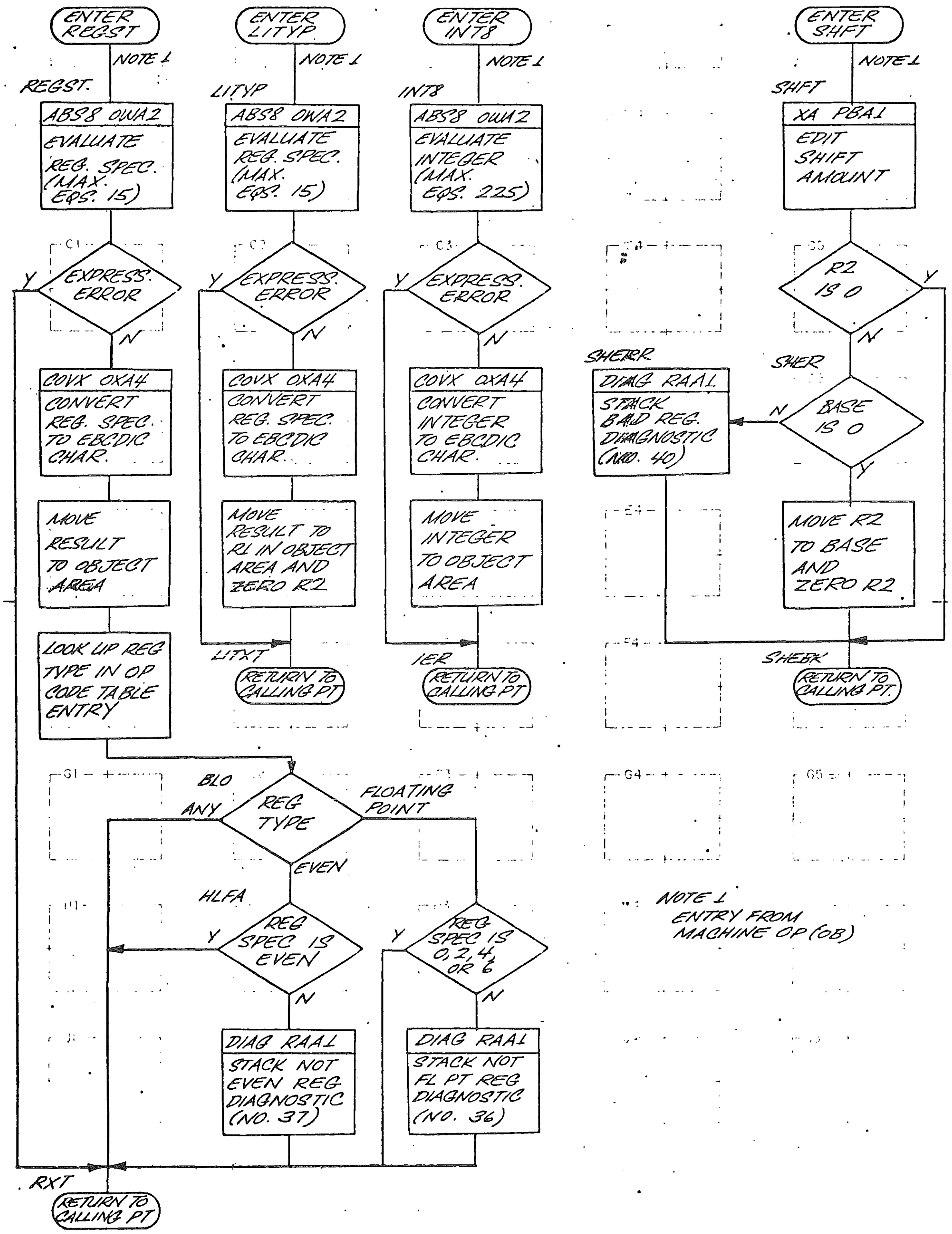


CHART OX. DOUB, HALF, CONVX SUBROUTINES



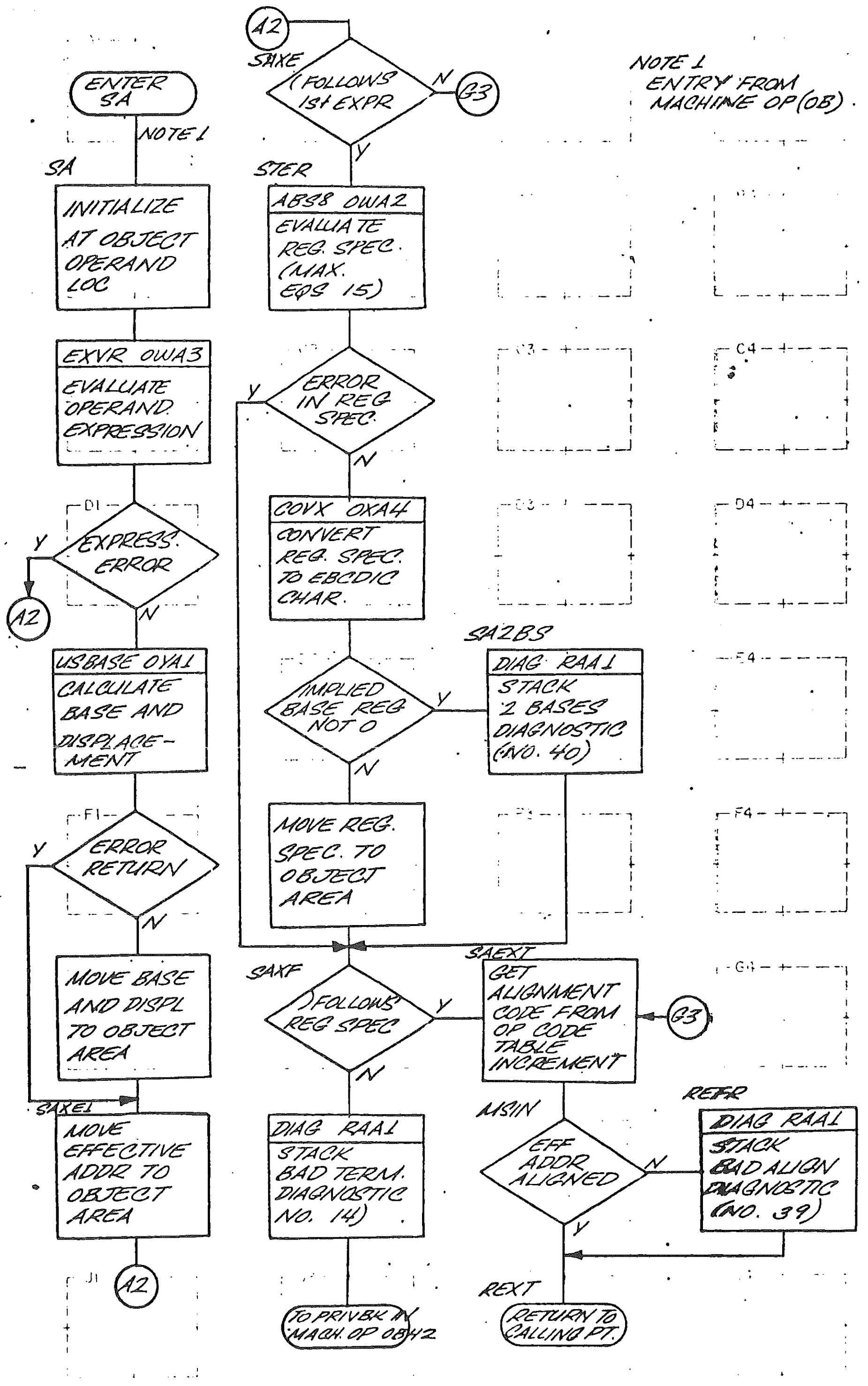
ENTER USBASE
NOTE 1
USBASE
COVX OXA4
CONVERT ADDR TO EBCDIC CHAR.
SAVE EFFECTIVE ADDR + SET UP FOR USING TBL SEARCH
A2
MLOOP
ENTRY AVAILABLE
Y
N
H2
RELOC IDS MATCH
ADDR. >ENTRY
ADDR. <ENTRY + 4096
COMPUTE DISPL WITH THIS BASE VALUE
DISPL. < PREVIOUS DISPL.
SAVE DISPL. AND BASE REG ADDR
H2
INCT
GO TO NEXT USING TABLE ENTRY
END OF TABLE
A3
BASE AND DISPL FOUND
COVX OXA4
CONVERT BASE AND DISPL TO EBCDIC CHAR.
NORMAL RETURN TO CALLING POINT PLUS 4
NOTINU
DIAG RAA1
STACK NO BASE DIAGNOSTIC (NO. 38)
ERR. RET. TO CALL. PT.
NOTE 1
ENTRY FROM PASS2
SBRTNS
SA PA
XA PB
SLA4Q PC
SLA PD

CHART OY. USBASE SUBROUTINE



ENTER REGST
NOTE 1
REGST
ABS8 OWA2
EVALUATE REG. SPEC. (MAX. EQS. 15)
C1
EXPRESS. ERROR
Y
N
COVX OXA4
CONVERT REG. SPEC. TO EBCDIC CHAR.
MOVE RESULT TO OBJECT AREA
LOOK UP REG TYPE IN OP CODE TABLE ENTRY
ENTER LITYP
NOTE 1
LITYP
ABS8 OWA2
EVALUATE REG. SPEC. (MAX. EQS. 15)
EXPRESS. ERROR
Y
N
COVX OXA4
CONVERT REG. SPEC. TO EBCDIC CHAR.
MOVE RESULT TO R1 IN OBJECT AREA AND ZERO R2
LITXT
RETURN TO CALLING PT
ENTER INT8
NOTE 1
INT8
ABS8 OWA2
EVALUATE INTEGER (MAX. EQS. 225)
C3
EXPRESS. ERROR
Y
N
COVX OXA4
CONVERT INTEGER TO EBCDIC CHAR.
MOVE INTEGER TO OBJECT AREA
IER
RETURN TO CALLING PT
ENTER SHFT
NOTE 1
SHFT
XA PBA1
EDIT SHIFT AMOUNT
R2 IS 0
Y
N
SHER
BASE IS 0
N
Y
SHERR
DIAG RAA1
STACK BAD REG. DIAGNOSTIC (NO. 40)
MOVE R2 TO BASE AND ZERO R2
SHEBK
RETURN TO CALLING PT.
BLO
REG TYPE
ANY
FLOATING POINT
EVEN
HLFA
REG SPEC IS EVEN
Y
N
DIAG RAA1
STACK NOT EVEN REG DIAGNOSTIC (NO. 37)
REG SPEC IS 0,2,4, OR 6
Y
N
DIAG RAA1
STACK NOT FL PT REG DIAGNOSTIC (NO. 36)
RXT
RETURN TO CALLING PT
NOTE 1
ENTRY FROM MACHINE OP (OB)

CHART OZ. REGST, LITYP, INT8, SHFT SUBROUTINES



A2
SAXE
(FOLLOWS 1st EXPR
N
G3
Y
NOTE 1 ENTRY FROM MACHINE OP (OB)
ENTER SA
NOTE 1
SA
INITIALIZE AT OBJECT OPERAND LOC
STER
ABS8 OWA2
EVALUATE REG. SPEC. (MAX. EQS 15)
EXVR OWA3
EVALUATE OPERAND EXPRESSION
Y
ERROR IN REG SPEC
N
Y
EXPRESS. ERROR
N
A2
COVX OXA4
CONVERT REG. SPEC. TO EBCDIC CHAR.
USBASE OYA1
CALCULATE BASE AND DISPLACE-MENT
SA2BS
IMPLIED BASE REG NOT 0
Y
N
DIAG RAA1
STACK 2 BASES DIAGNOSTIC (NO. 40)
Y
ERROR RETURN
N
MOVE REG. SPEC. TO OBJECT AREA
MOVE BASE AND DISPL TO OBJECT AREA
SAXF
) FOLLOWS REG SPEC
Y
N
SAEXT
GET ALIGNMENT CODE FROM OP CODE TABLE INCREMENT
G3
SAXE1
MOVE EFFECTIVE ADDR TO OBJECT AREA
DIAG RAA1
STACK BAD TERM. DIAGNOSTIC NO. 14)
MSIN
EFF ADDR ALIGNED
N
Y
REFR
DIAG RAA1
STACK BAD ALIGN DIAGNOSTIC (NO. 39)
A2
REXT
TO PRIVBK IN MACH. OP OB42
RETURN TO CALLING PT.

CHART PA. SA SUBROUTINE

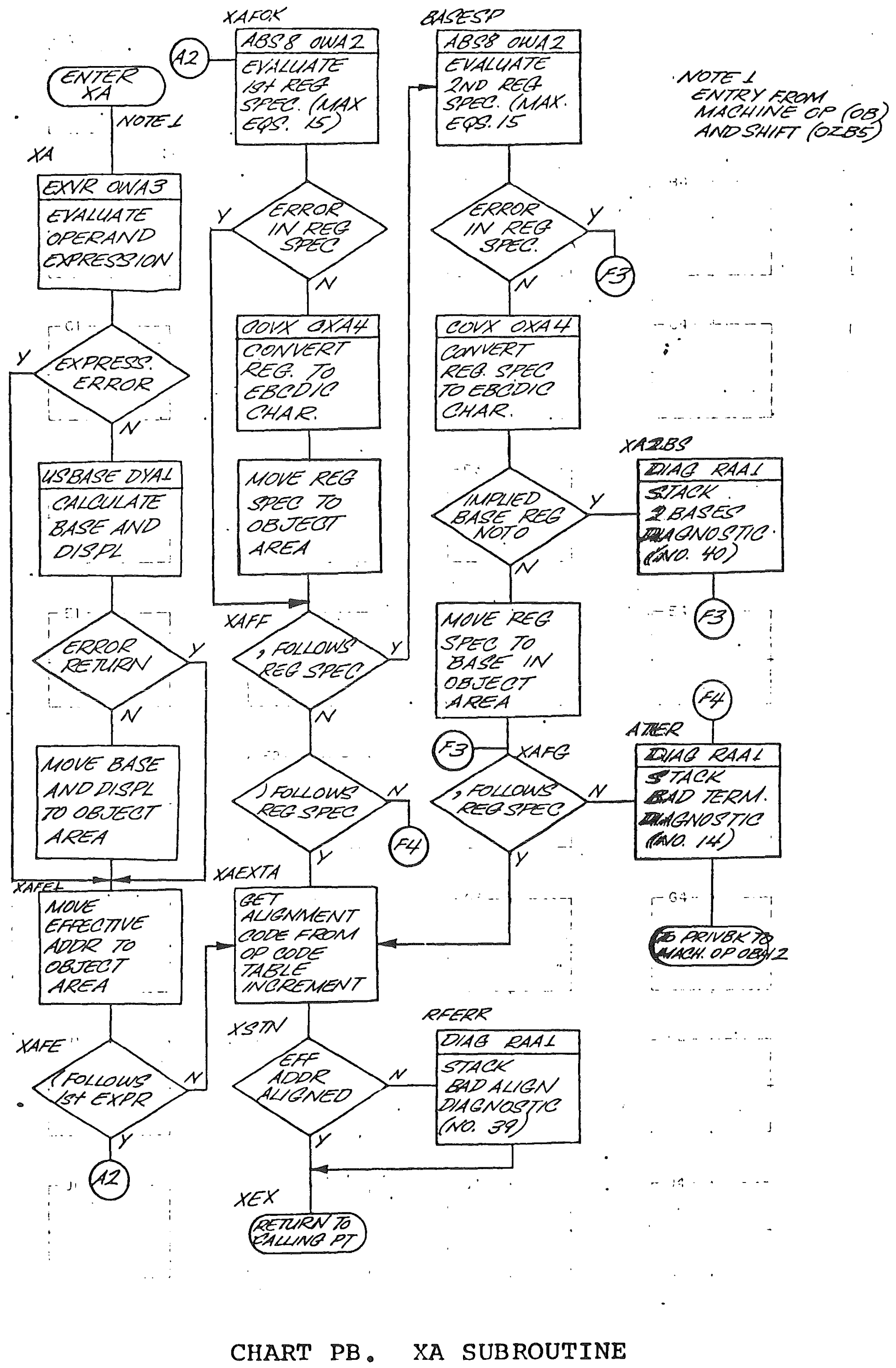
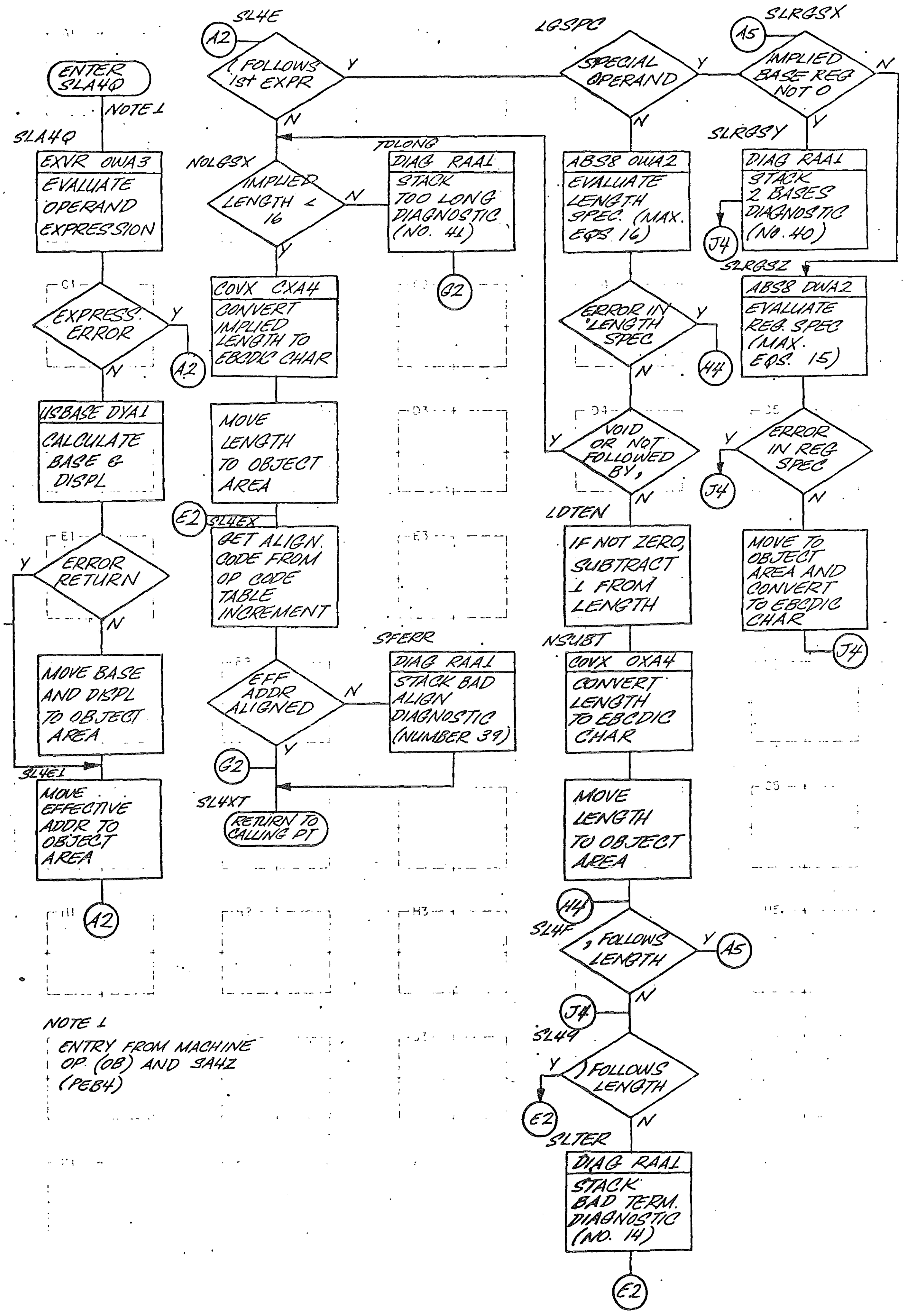

ENTER XA
NOTE 1
XA
EXVR OWA3
EVALUATE OPERAND EXPRESSION
EXPRESS. ERROR
Y
N
USBASE DYA1
CALCULATE BASE AND DISPL
ERROR RETURN
Y
N
MOVE BASE AND DISPL TO OBJECT AREA
XAFEL
MOVE EFFECTIVE ADDR TO OBJECT AREA
XAFE
(FOLLOWS 1st EXPR
N
Y
A2
XAFOK
ABS8 OWA2
EVALUATE 1st REG SPEC. (MAX EQS. 15)
ERROR IN REG SPEC
Y
N
COVX OXA4
CONVERT REG. TO EBCDIC CHAR.
MOVE REG SPEC TO OBJECT AREA
XAFF
, FOLLOWS REG SPEC
Y
N
) FOLLOWS REG SPEC
N
F4
Y
XAEXTA
GET ALIGNMENT CODE FROM OP CODE TABLE INCREMENT
XSTN
EFF ADDR ALIGNED
N
Y
XEX
RETURN TO CALLING PT
BASESP
ABS8 OWA2
EVALUATE 2ND REG SPEC. (MAX EQS. 15
ERROR IN REG SPEC.
Y
F3
N
COVX OXA4
CONVERT REG. SPEC TO EBCDIC CHAR.
IMPLIED BASE REG NOT 0
Y
N
MOVE REG SPEC TO BASE IN OBJECT AREA
F3
XAFG
, FOLLOWS REG SPEC
N
Y
RFERR
DIAG RAA1
STACK BAD ALIGN DIAGNOSTIC (NO. 39)
XA2BS
DIAG RAA1
STACK 2 BASES DIAGNOSTIC (NO. 40)
F3
F4
ATER
DIAG RAA1
STACK BAD TERM. DIAGNOSTIC (NO. 14)
TO PRIVBK TO MACH. OP OBA2
NOTE 1
ENTRY FROM MACHINE OP (OB) AND SHIFT (OZB5)

CHART PB. XA SUBROUTINE



ENTER SLA4Q
NOTE 1
SLA4Q
EXVR OWA3
EVALUATE OPERAND EXPRESSION
EXPRESS. ERROR
Y
N
A2
USBASE DYA1
CALCULATE BASE & DISPL
ERROR RETURN
MOVE BASE AND DISPL TO OBJECT AREA
SL4E1
MOVE EFFECTIVE ADDR TO OBJECT AREA
A2
SL4E
(FOLLOWS 1st EXPR
NOLGSX
IMPLIED LENGTH < 16
TOLONG
DIAG RAA1
STACK TOO LONG DIAGNOSTIC (NO. 41)
G2
COVX CXA4
CONVERT IMPLIED LENGTH TO EBCDIC CHAR
MOVE LENGTH TO OBJECT AREA
E2
SL4EX
GET ALIGN. CODE FROM OP CODE TABLE INCREMENT
SFERR
EFF ADDR ALIGNED
DIAG RAA1
STACK BAD ALIGN DIAGNOSTIC (NUMBER 39)
SL4XT
RETURN TO CALLING PT
LGSPC
SPECIAL OPERAND
ABS8 OWA2
EVALUATE LENGTH SPEC (MAX. EQS 16)
ERROR IN LENGTH SPEC
H4
VOID OR NOT FOLLOWED BY ,
LDTEN
IF NOT ZERO, SUBTRACT 1 FROM LENGTH
NSUBT
COVX OXA4
CONVERT LENGTH TO EBCDIC CHAR
MOVE LENGTH TO OBJECT AREA
SL4F
, FOLLOWS LENGTH
J4
SL49
) FOLLOWS LENGTH
SLTER
DIAG RAA1
STACK BAD TERM. DIAGNOSTIC (NO. 14)
SLRGSX
A5
IMPLIED BASE REG NOT 0
SLRGSY
DIAG RAA1
STACK 2 BASES DIAGNOSTIC (NO. 40)
SLRGSZ
ABS8 DWA2
EVALUATE REG. SPEC (MAX. EQS. 15)
ERROR IN REG SPEC
MOVE TO OBJECT AREA AND CONVERT TO EBCDIC CHAR
NOTE 1
ENTRY FROM MACHINE OP (OB) AND SA4Z (PEB4)

CHART PC. SLA4Q SUBROUTINE

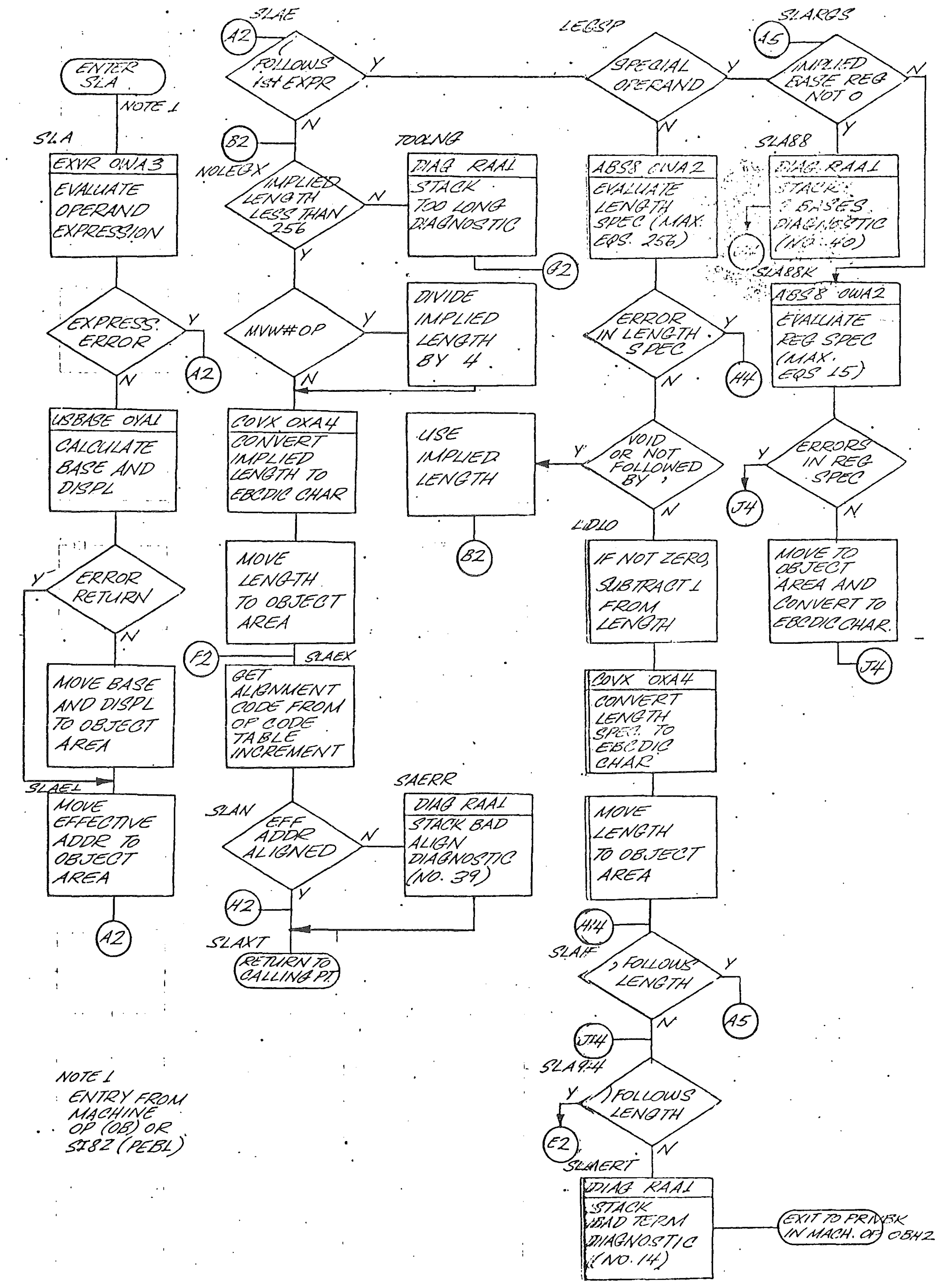

ENTER SLA
NOTE 1
SLA
EXVR OWA3
EVALUATE OPERAND EXPRESSION
EXPRESS. ERROR
Y
N
A2
USBASE OYA1
CALCULATE BASE AND DISPL
ERROR RETURN
MOVE BASE AND DISPL TO OBJECT AREA
SLAE1
MOVE EFFECTIVE ADDR TO OBJECT AREA
SLAE
(FOLLOWS 1st EXPR
B2
NOLEGX
IMPLIED LENGTH LESS THAN 256
TOOLNG
DIAG RAAL
STACK TOO LONG DIAGNOSTIC
F2
MVW#OP
DIVIDE IMPLIED LENGTH BY 4
COVX OXA4
CONVERT IMPLIED LENGTH TO EBCDIC CHAR
MOVE LENGTH TO OBJECT AREA
SLAEX
GET ALIGNMENT CODE FROM OP CODE TABLE INCREMENT
SLAN
EFF ADDR ALIGNED
SAERR
DIAG RAAL
STACK BAD ALIGN DIAGNOSTIC (NO. 39)
H2
SLAXT
RETURN TO CALLING PT.
LEGSP
SPECIAL OPERAND
ABSS OWA2
EVALUATE LENGTH SPEC (MAX. EQS 256)
ERROR IN LENGTH SPEC
H4
VOID OR NOT FOLLOWED BY ,
USE IMPLIED LENGTH
LDLO
IF NOT ZERO, SUBTRACT 1 FROM LENGTH
COVX OXA4
CONVERT LENGTH SPEC TO EBCDIC CHAR
MOVE LENGTH TO OBJECT AREA
SLAIF
, FOLLOWS LENGTH
A5
J4
SLA94
) FOLLOWS LENGTH
E2
SLAERT
DIAG RAAL
STACK BAD TERM DIAGNOSTIC (NO. 14)
EXIT TO PRNBK IN MACH. OP OBH2
SLARGS
IMPLIED BASE REG NOT 0
SLA88
DIAG RAAL
STACK 2 BASES DIAGNOSTIC (NO. 40)
SLA88K
ABS8 OWA2
EVALUATE REG SPEC (MAX. EQS 15)
ERRORS IN REG SPEC
MOVE TO OBJECT AREA AND CONVERT TO EBCDIC CHAR.
NOTE 1
ENTRY FROM MACHINE OP (OB) OR SI82 (PEBL)

CHART PD. SLA SUBROUTINE

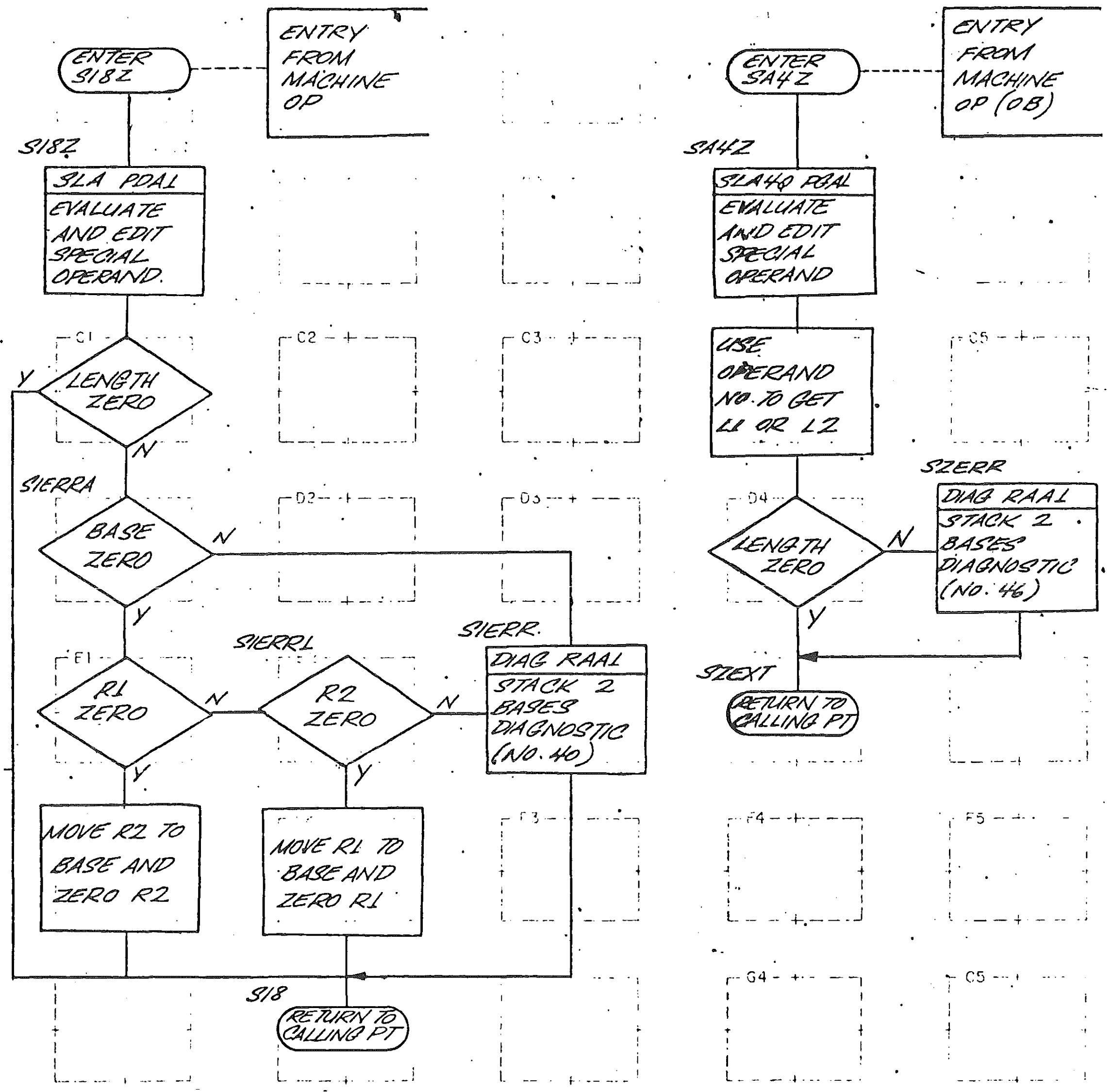

ENTER SI8Z
ENTRY FROM MACHINE OP
SI8Z
SLA PDAL
EVALUATE AND EDIT SPECIAL OPERAND.
C1
C2
C3
Y
LENGTH ZERO
N
SIERRA
D2
D3
BASE ZERO
N
Y
SIERRL
SIERR
DIAG RAAL
STACK 2 BASES DIAGNOSTIC (NO. 40)
E1
R1 ZERO
N
R2 ZERO
N
Y
Y
MOVE R2 TO BASE AND ZERO R2
MOVE R1 TO BASE AND ZERO R1
F3
SI8
RETURN TO CALLING PT
ENTER SA4Z
ENTRY FROM MACHINE OP (0B)
SA4Z
SLA40 PDAL
EVALUATE AND EDIT SPECIAL OPERAND
USE OPERAND NO. TO GET L1 OR L2
C5
SZERR
D4
DIAG RAAL
STACK 2 BASES DIAGNOSTIC (NO. 46)
LENGTH ZERO
N
Y
SZEXT
RETURN TO CALLING PT
F4
F5
G4
G5

CHART PE. SI8Z, SA4Z SUBROUTINES

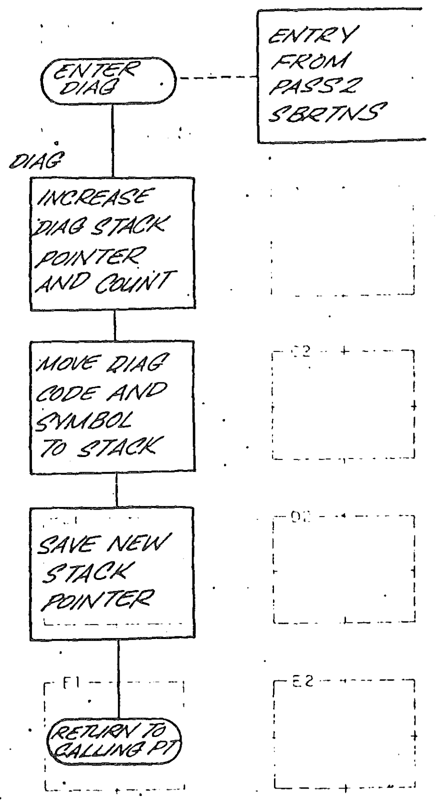


CHART RA. DIAG ROUTINE

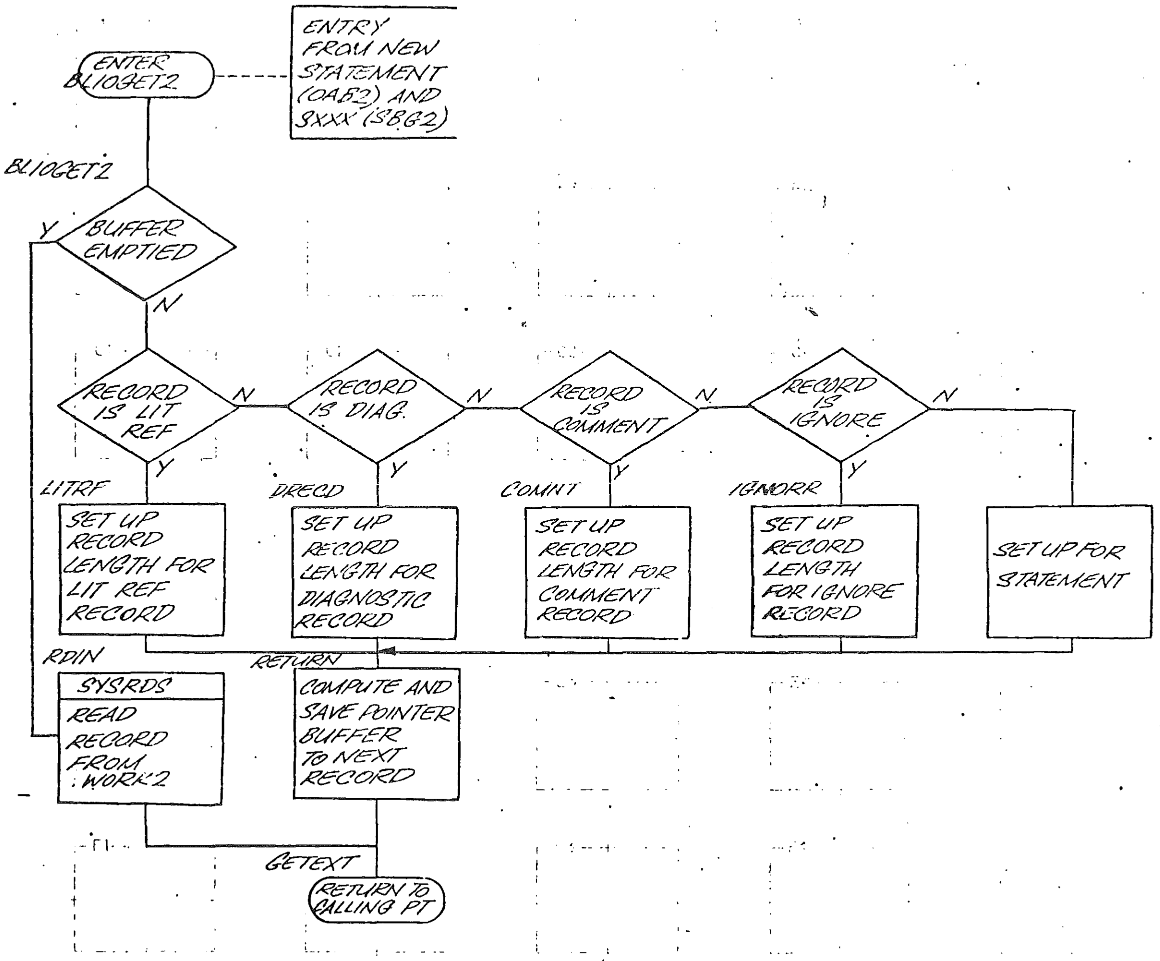

ENTER BLIOGET2
ENTRY FROM NEW STATEMENT (OAB2) AND SXXX (SBG2)
BLIOGET2
BUFFER EMPTIED
Y
N
RECORD IS LIT REF
N
RECORD IS DIAG.
N
RECORD IS COMMENT
N
RECORD IS IGNORE
N
Y
LITRF
SET UP RECORD LENGTH FOR LIT REF RECORD
Y
DRECD
SET UP RECORD LENGTH FOR DIAGNOSTIC RECORD
Y
COMNT
SET UP RECORD LENGTH FOR COMMENT RECORD
Y
IGNORR
SET UP RECORD LENGTH FOR IGNORE RECORD
SET UP FOR STATEMENT
RDIN
SYSRDS
READ RECORD FROM :WORK2
RETURN
COMPUTE AND SAVE POINTER BUFFER TO NEXT RECORD
GETEXT
RETURN TO CALLING PT

CHART QA. BLIOGET2 ROUTINE

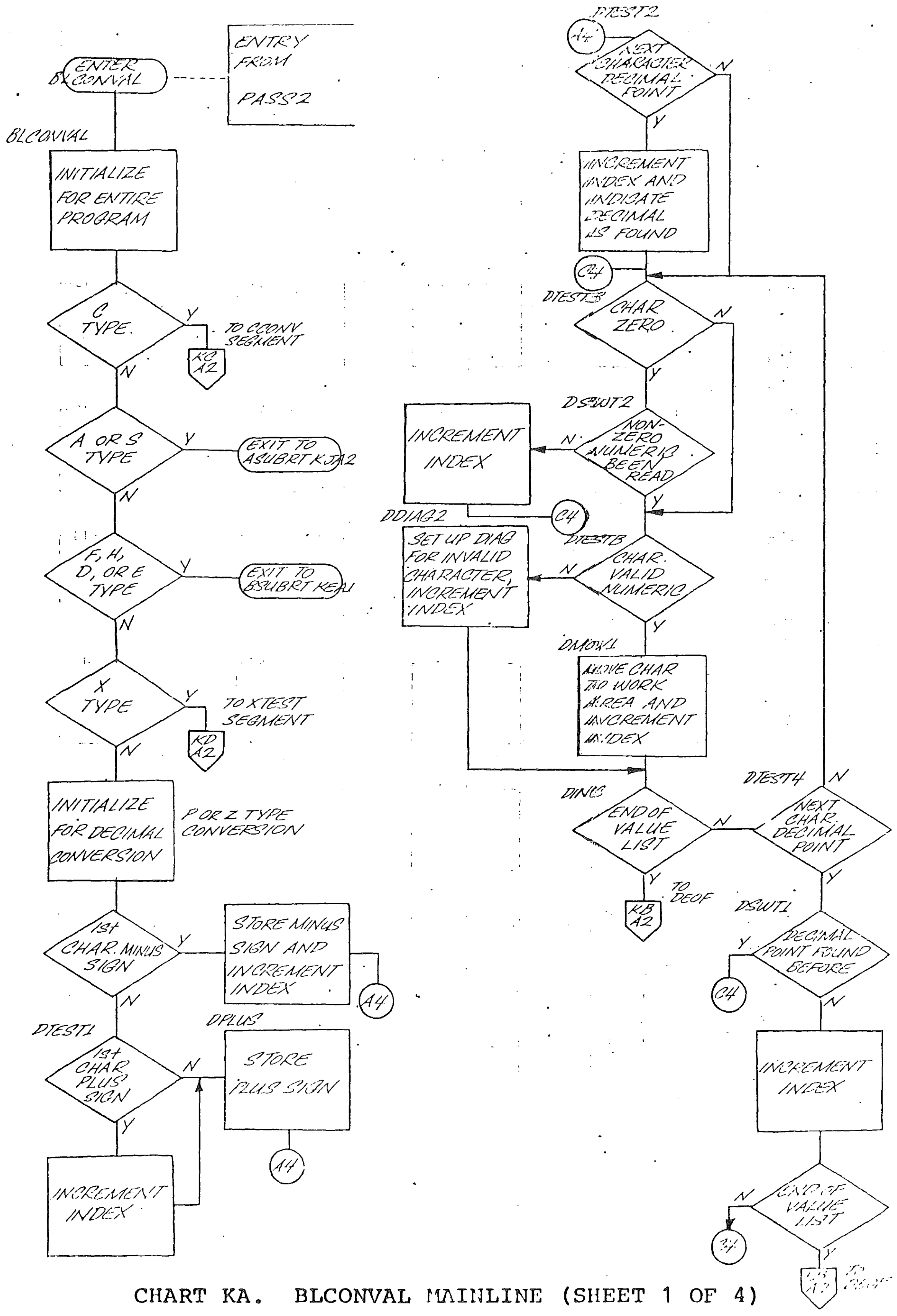


CHART KA. BLCONVAL MAINLINE (SHEET 1 OF 4)

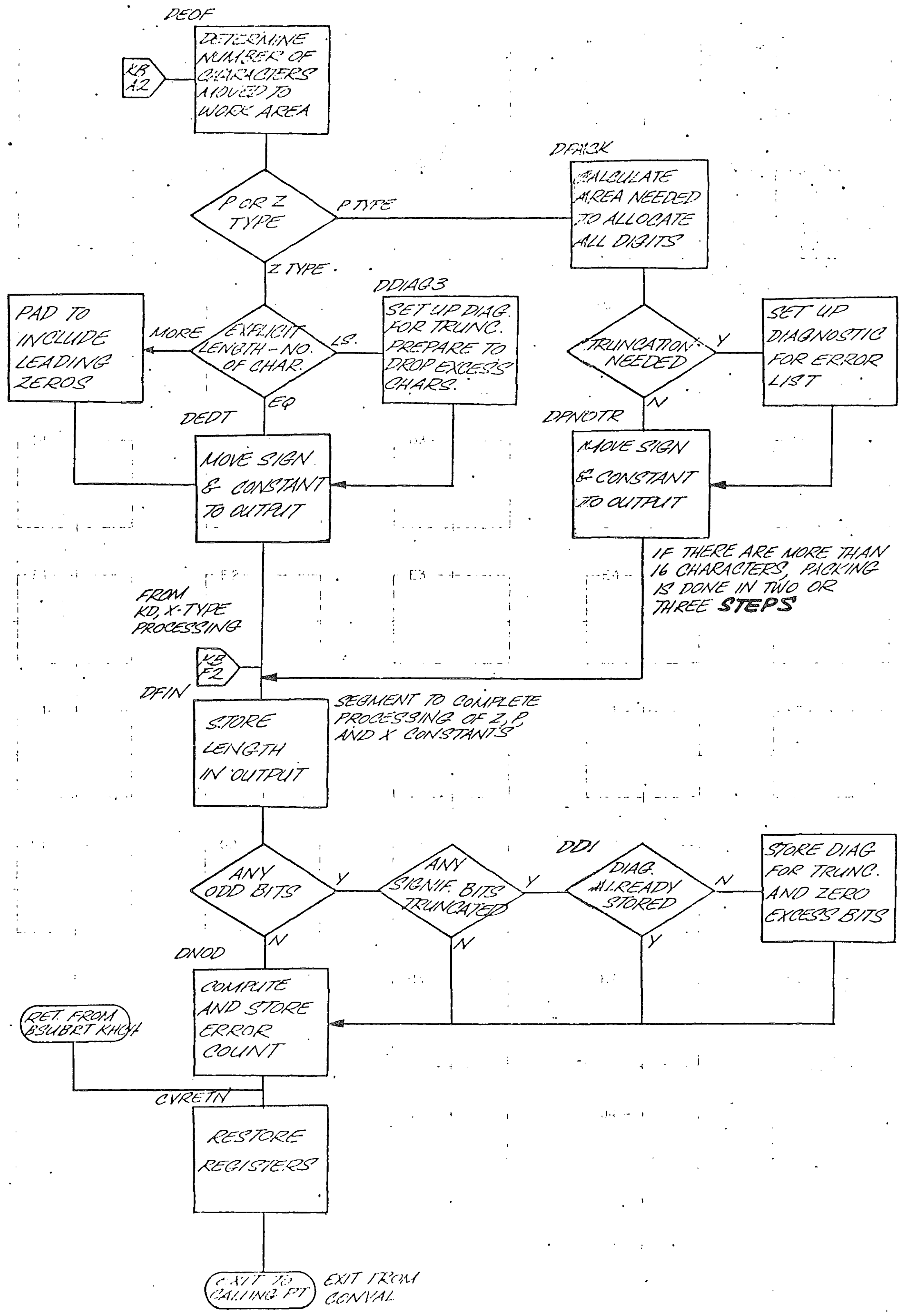

DEOF
KB A2
DETERMINE NUMBER OF CHARACTERS MOVED TO WORK AREA
P OR Z TYPE
P TYPE
DPACK
CALCULATE AREA NEEDED TO ALLOCATE ALL DIGITS
Z TYPE
DDIAG3
PAD TO INCLUDE LEADING ZEROS
MORE
EXPLICIT LENGTH - NO. OF CHAR.
LS.
SET UP DIAG. FOR TRUNC. PREPARE TO DROP EXCESS CHARS.
TRUNCATION NEEDED
Y
SET UP DIAGNOSTIC FOR ERROR LIST
EQ
N
DEDT
MOVE SIGN & CONSTANT TO OUTPUT
DPNOTR
MOVE SIGN & CONSTANT TO OUTPUT
IF THERE ARE MORE THAN 16 CHARACTERS, PACKING IS DONE IN TWO OR THREE STEPS
FROM KD, X-TYPE PROCESSING
KB F2
DFIN
STORE LENGTH IN OUTPUT
SEGMENT TO COMPLETE PROCESSING OF Z, P AND X CONSTANTS
DDI
ANY ODD BITS
Y
ANY SIGNIF. BITS TRUNCATED
Y
DIAG. ALREADY STORED
N
STORE DIAG FOR TRUNC. AND ZERO EXCESS BITS
N
N
Y
DNOD
COMPUTE AND STORE ERROR COUNT
RET. FROM BSUBRT KH04
CVRETN
RESTORE REGISTERS
EXIT TO CALLING PT
EXIT FROM CONVAL

CHART KB. BLCONVAL MAINLINE (SHEET 2 OF 4)

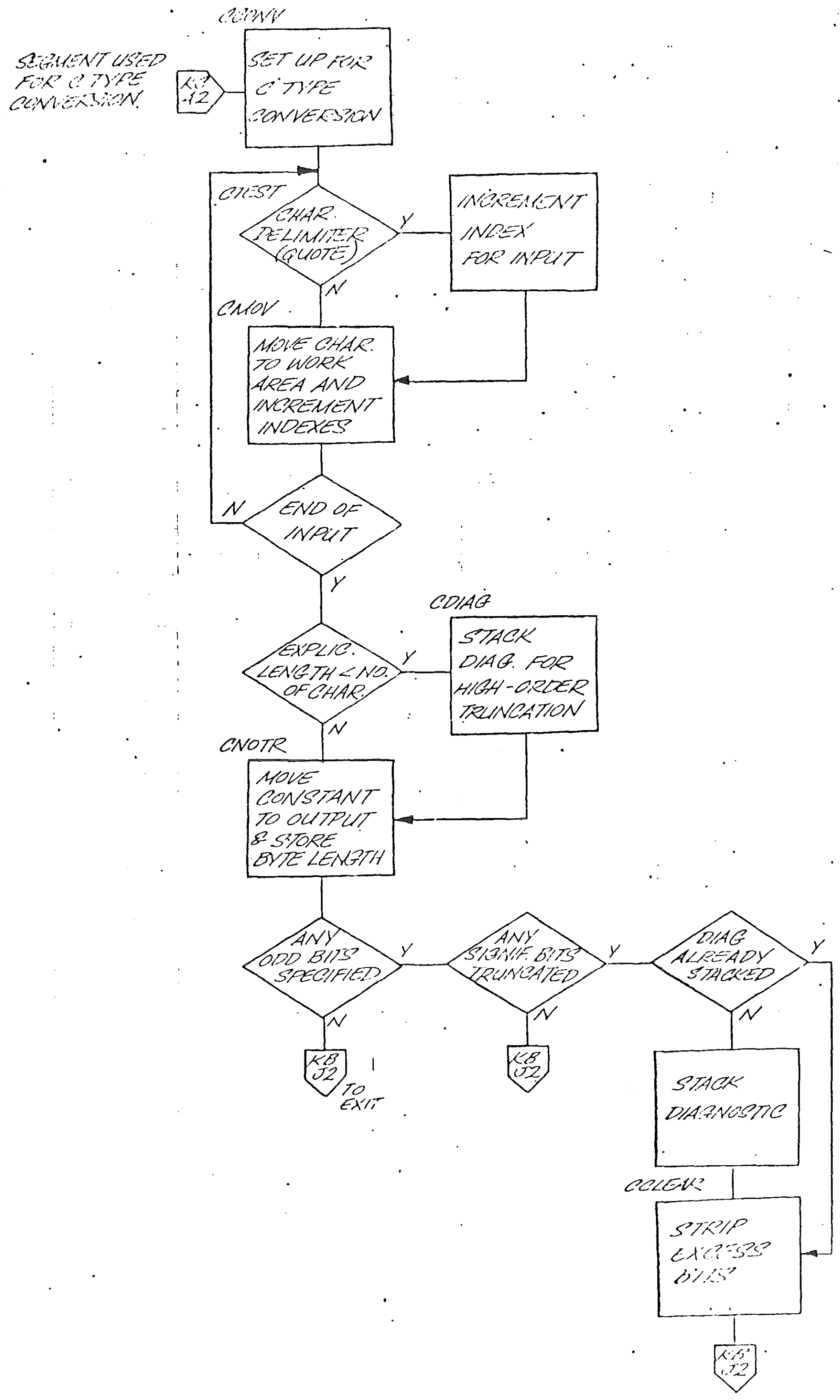

CCONV
SEGMENT USED FOR C TYPE CONVERSION.
K3 A2
SET UP FOR C TYPE CONVERSION
CTEST
CHAR. DELIMITER (QUOTE)
Y
N
INCREMENT INDEX FOR INPUT
CMOV
MOVE CHAR. TO WORK AREA AND INCREMENT INDEXES
END OF INPUT
N
Y
CDIAG
EXPLIC. LENGTH < NO. OF CHAR.
Y
N
STACK DIAG. FOR HIGH-ORDER TRUNCATION
CNOTR
MOVE CONSTANT TO OUTPUT & STORE BYTE LENGTH
ANY ODD BITS SPECIFIED
Y
N
K8 J2
TO EXIT
ANY SIGNIF. BITS TRUNCATED
Y
N
K8 J2
DIAG ALREADY STACKED
Y
N
STACK DIAGNOSTIC
CCLEAR
STRIP EXCESS BITS
K8 J2

CHART KC. BLCONVAL MAINLINE (SHEET 3 OF 4)

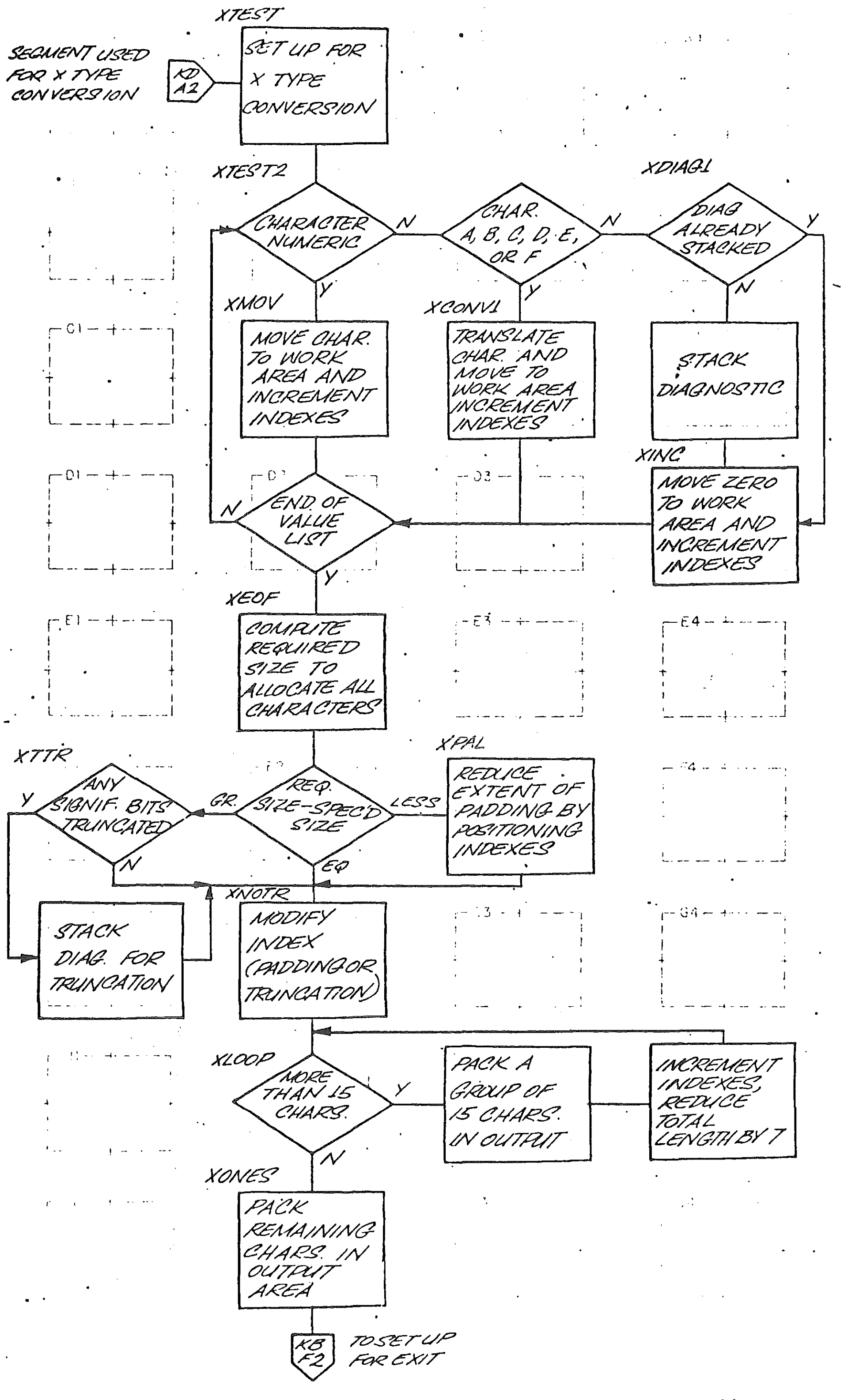

XTEST
SEGMENT USED FOR X TYPE CONVERSION
KD A2
SET UP FOR X TYPE CONVERSION
XTEST2
CHARACTER NUMERIC
N
Y
CHAR. A, B, C, D, E, OR F
N
Y
XDIAG1
DIAG ALREADY STACKED
Y
N
XMOV
MOVE CHAR. TO WORK AREA AND INCREMENT INDEXES
XCONV1
TRANSLATE CHAR. AND MOVE TO WORK AREA INCREMENT INDEXES
STACK DIAGNOSTIC
XINC
MOVE ZERO TO WORK AREA AND INCREMENT INDEXES
END OF VALUE LIST
N
Y
XEOF
COMPUTE REQUIRED SIZE TO ALLOCATE ALL CHARACTERS
XTTR
ANY SIGNIF. BITS TRUNCATED
Y
N
GR.
REQ. SIZE-SPEC'D SIZE
LESS
EQ
XPAL
REDUCE EXTENT OF PADDING BY POSITIONING INDEXES
STACK DIAG. FOR TRUNCATION
XNOTR
MODIFY INDEX (PADDING OR TRUNCATION)
XLOOP
MORE THAN 15 CHARS.
Y
N
PACK A GROUP OF 15 CHARS. IN OUTPUT
INCREMENT INDEXES, REDUCE TOTAL LENGTH BY 7
XONES
PACK REMAINING CHARS. IN OUTPUT AREA
KB F2
TO SET UP FOR EXIT

CHART KD. BLCONVAL MAINLINE (SHEET 4 OF 4)

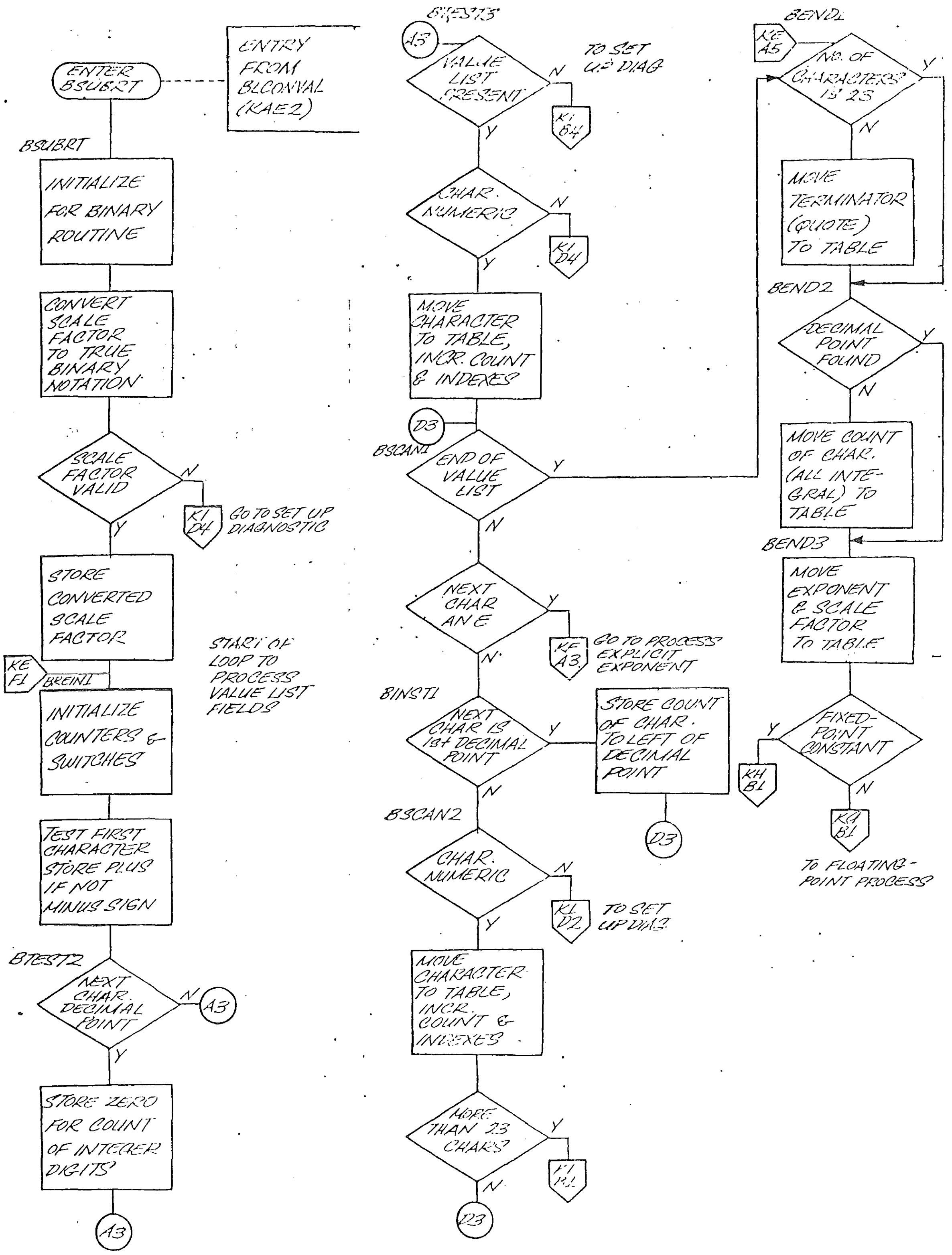

ENTER BSUBRT
ENTRY FROM BLCONVAL (KAE2)
BSUBRT
INITIALIZE FOR BINARY ROUTINE
CONVERT SCALE FACTOR TO TRUE BINARY NOTATION
SCALE FACTOR VALID
N
Y
KI D4
GO TO SET UP DIAGNOSTIC
STORE CONVERTED SCALE FACTOR
START OF LOOP TO PROCESS VALUE LIST FIELDS
KE F1
BREIN1
INITIALIZE COUNTERS & SWITCHES
TEST FIRST CHARACTER STORE PLUS IF NOT MINUS SIGN
BTEST2
NEXT CHAR. DECIMAL POINT
N
A3
Y
STORE ZERO FOR COUNT OF INTEGER DIGITS
A3
BTEST3
A3
VALUE LIST PRESENT
N
TO SET UP DIAG
KI B4
Y
CHAR. NUMERIC
N
KI D4
Y
MOVE CHARACTER TO TABLE, INCR. COUNT & INDEXES
D3
BSCAN1
END OF VALUE LIST
Y
N
NEXT CHAR AN E
Y
KE A3
GO TO PROCESS EXPLICIT EXPONENT
N
BINST1
NEXT CHAR IS 1st DECIMAL POINT
Y
STORE COUNT OF CHAR. TO LEFT OF DECIMAL POINT
D3
N
BSCAN2
CHAR. NUMERIC
N
KI D2
TO SET UP DIAG
Y
MOVE CHARACTER TO TABLE, INCR. COUNT & INDEXES
MORE THAN 23 CHARS?
Y
FI B1
N
D3
BEND1
KE A5
NO. OF CHARACTERS IS 23
Y
N
MOVE TERMINATOR (QUOTE) TO TABLE
BEND2
DECIMAL POINT FOUND
Y
N
MOVE COUNT OF CHAR. (ALL INTEGRAL) TO TABLE
BEND3
MOVE EXPONENT & SCALE FACTOR TO TABLE
FIXED-POINT CONSTANT
Y
KH B1
N
KG B1
TO FLOATING-POINT PROCESS

CHART KE. BSUBRT ROUTINE (SHEET 1 OF 5)

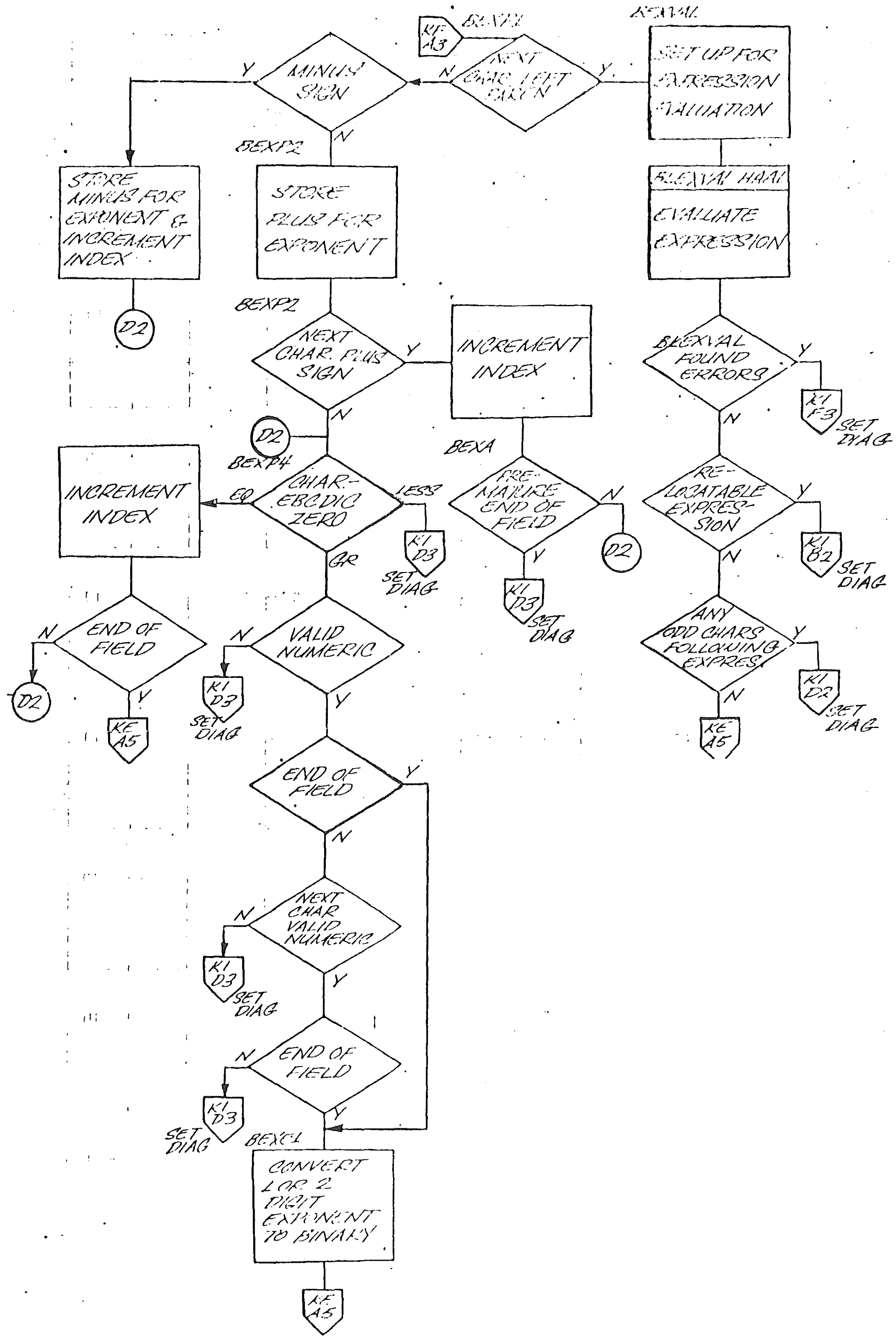


CHART KF. BSUBRT ROUTINE (SHEET 2 OF 5)

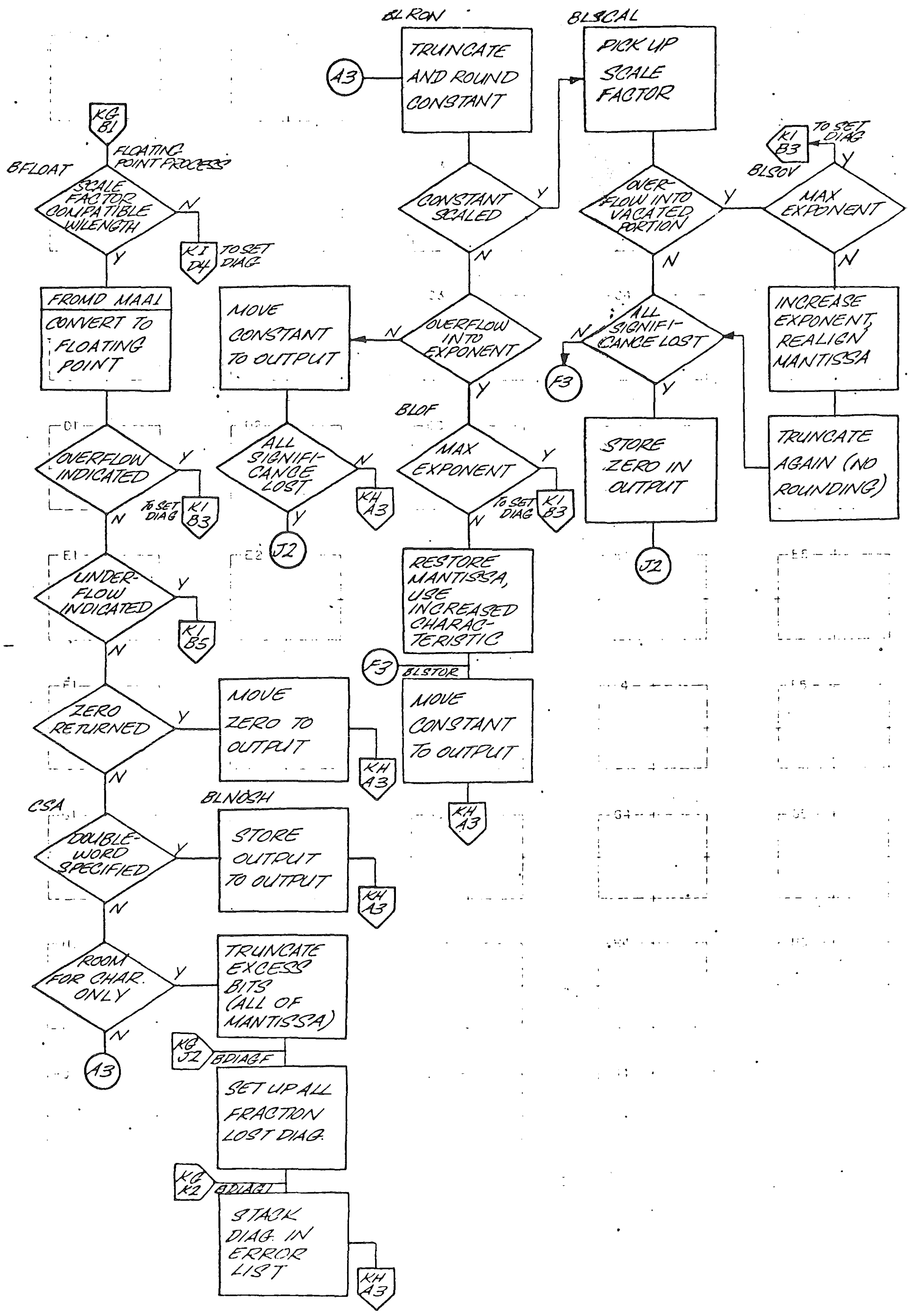


CHART KG. BSUBRT ROUTINE (SHEET 3 OF 5)

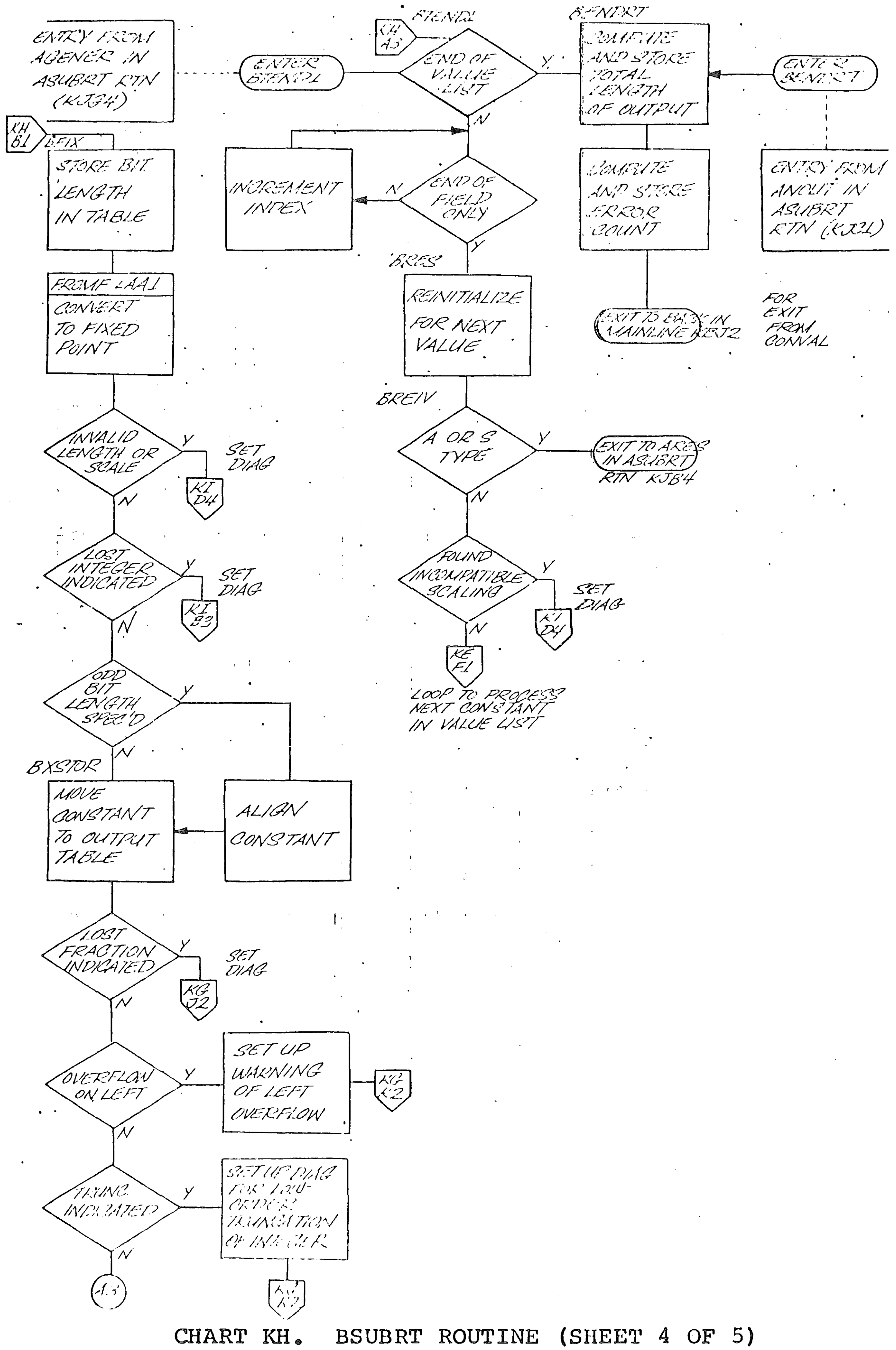

ENTRY FROM AGENER IN ASUBRT RTN (KJG4)
ENTER BTENDL
BTENDL
KH A3
END OF VALUE LIST
Y
N
BENDRT
COMPUTE AND STORE TOTAL LENGTH OF OUTPUT
ENTER BENDRT
ENTRY FROM ANOUT IN ASUBRT RTN (KJ1)
KH B1
BFIX
STORE BIT LENGTH IN TABLE
INCREMENT INDEX
N
END OF FIELD ONLY
Y
COMPUTE AND STORE ERROR COUNT
BRES
REINITIALIZE FOR NEXT VALUE
EXIT TO BASX IN MAINLINE KBJ2
FOR EXIT FROM CONVAL
FROMF LAA1
CONVERT TO FIXED POINT
BREIV
A OR S TYPE
Y
EXIT TO ARES IN ASUBRT RTN KJB4
INVALID LENGTH OR SCALE
Y
SET DIAG
KI D4
N
N
FOUND INCOMPATIBLE SCALING
Y
SET DIAG
KI D4
N
KE F1
LOOP TO PROCESS NEXT CONSTANT IN VALUE LIST
LOST INTEGER INDICATED
Y
SET DIAG
KI B3
N
ODD BIT LENGTH SPEC'D
Y
N
BXSTOR
MOVE CONSTANT TO OUTPUT TABLE
ALIGN CONSTANT
LOST FRACTION INDICATED
Y
SET DIAG
KG J2
N
OVERFLOW ON LEFT
Y
SET UP WARNING OF LEFT OVERFLOW
KG K2
N
TRUNC INDICATED
Y
SET UP DIAG FOR LOW-ORDER TRUNCATION OF INTEGER
N
A3
KG K2

CHART KH. BSUBRT ROUTINE (SHEET 4 OF 5)

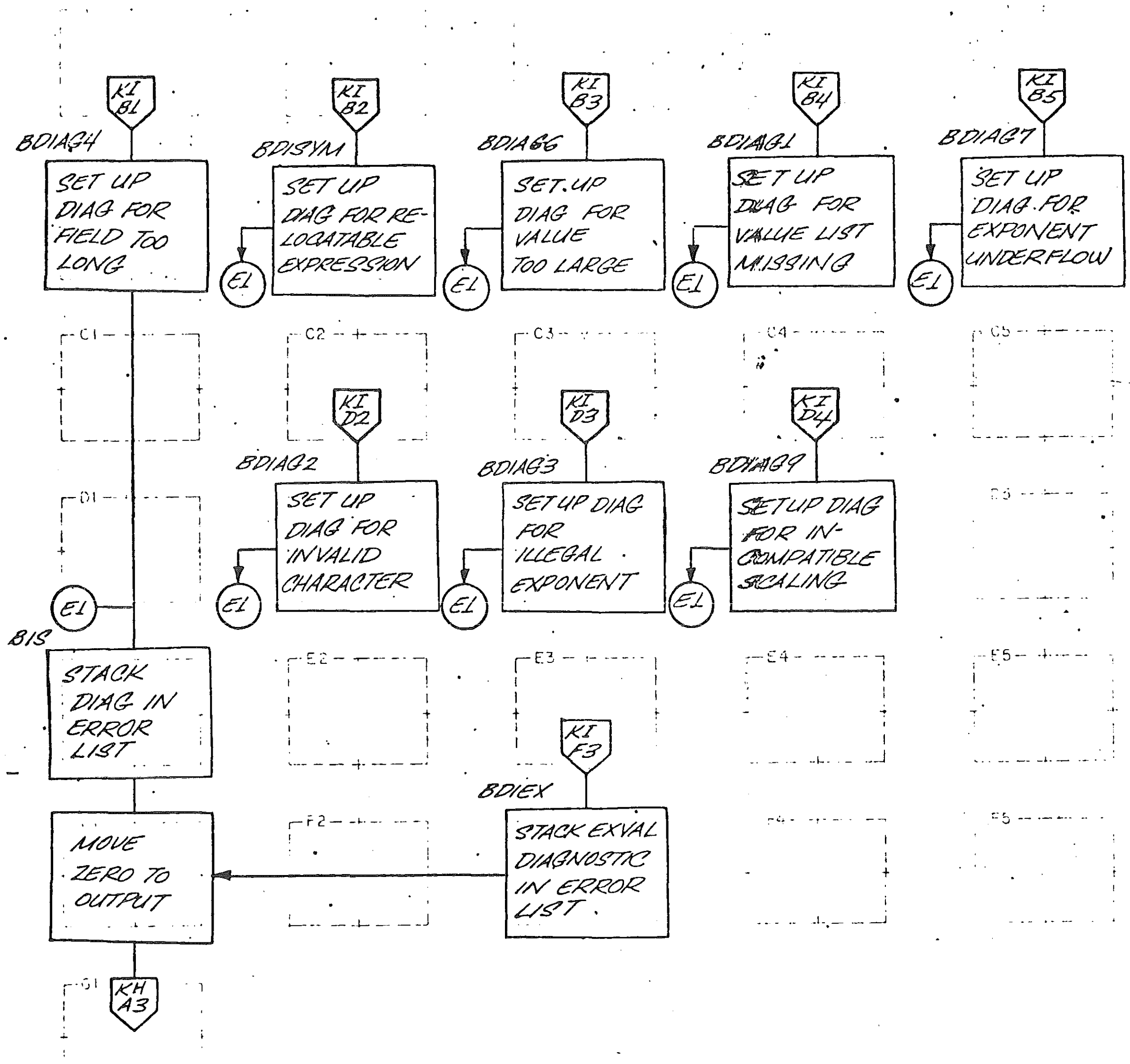

KI B1
BDIAG4
SET UP DIAG FOR FIELD TOO LONG
KI B2
BDISYM
SET UP DIAG FOR RE-LOCATABLE EXPRESSION
KI B3
BDIAG6
SET.UP DIAG FOR VALUE TOO LARGE
KI B4
BDIAG1
SET UP DIAG FOR VALUE LIST MISSING
KI B5
BDIAG7
SET UP DIAG.FOR EXPONENT UNDERFLOW
E1
KI D2
BDIAG2
SET UP DIAG FOR INVALID CHARACTER
KI D3
BDIAG3
SET UP DIAG FOR ILLEGAL EXPONENT
KI D4
BDIAG9
SET UP DIAG FOR IN-COMPATIBLE SCALING
BIS
STACK DIAG IN ERROR LIST
KI F3
BDIEX
STACK EXVAL DIAGNOSTIC IN ERROR LIST
MOVE ZERO TO OUTPUT
KH A3

CHART KI. BSUBRT ROUTINE (SHEET 5 OF 5)

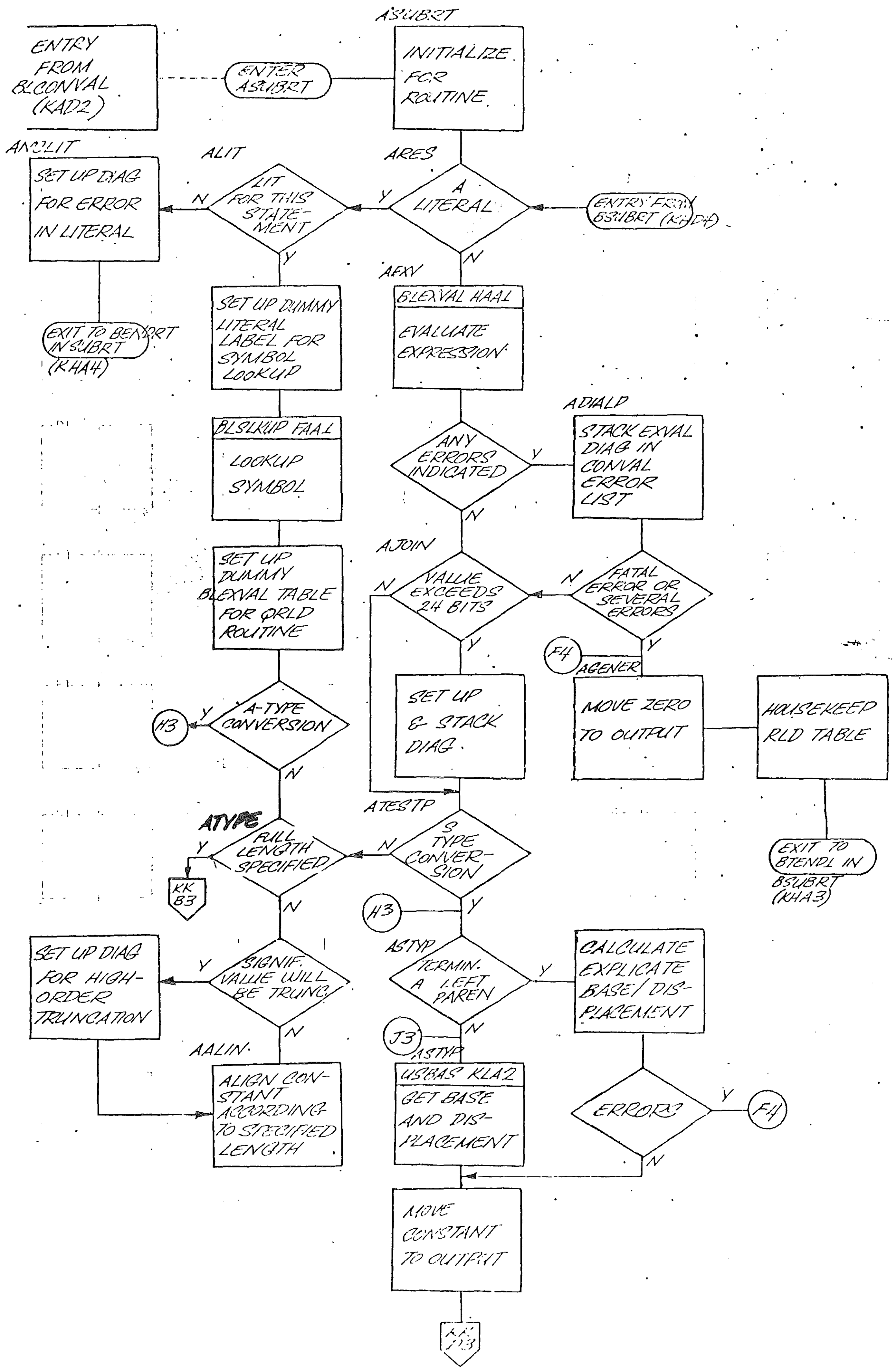


CHART KJ. ASUBRT ROUTINE (SHEET 1 OF 2)

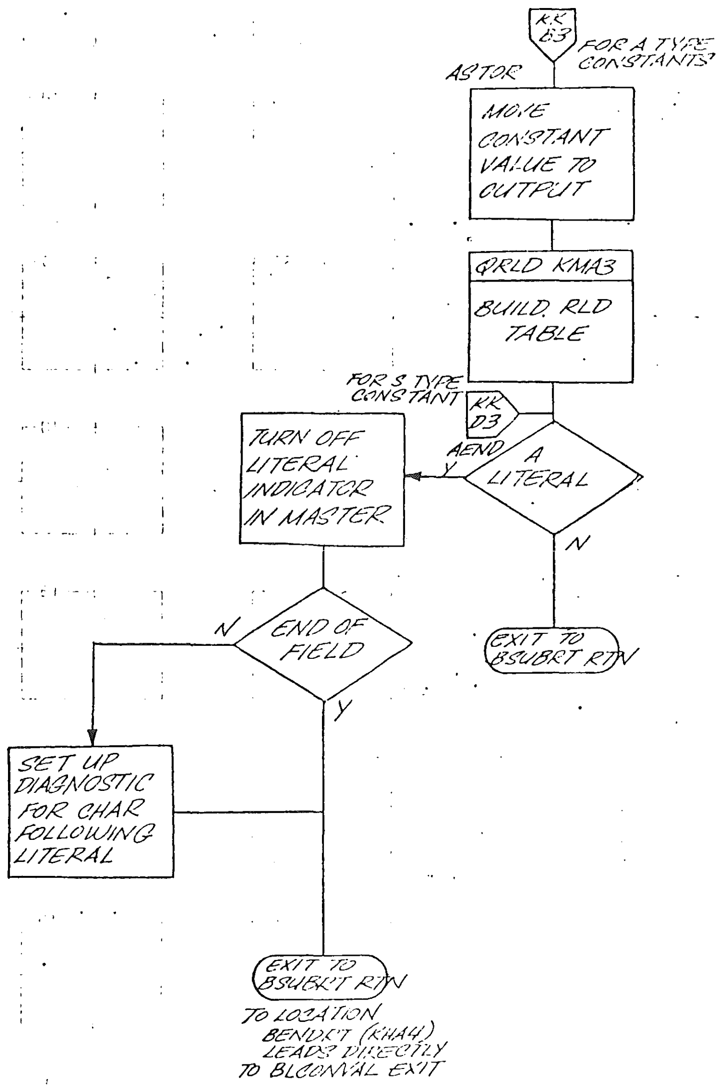

KK E3
FOR A TYPE CONSTANTS
ASTOR
MOVE CONSTANT VALUE TO OUTPUT
QRLD KMA3
BUILD RLD TABLE
FOR S TYPE CONSTANT
KK D3
AEND
A LITERAL
Y
N
TURN OFF LITERAL INDICATOR IN MASTER
EXIT TO BSUBRT RTN
END OF FIELD
N
Y
SET UP DIAGNOSTIC FOR CHAR FOLLOWING LITERAL
EXIT TO BSUBRT RTN
TO LOCATION BENDST (KHA4) LEADS DIRECTLY TO BLCONVAL EXIT

CHART KK. ASUBRT ROUTINE (SHEET 2 OF 2)

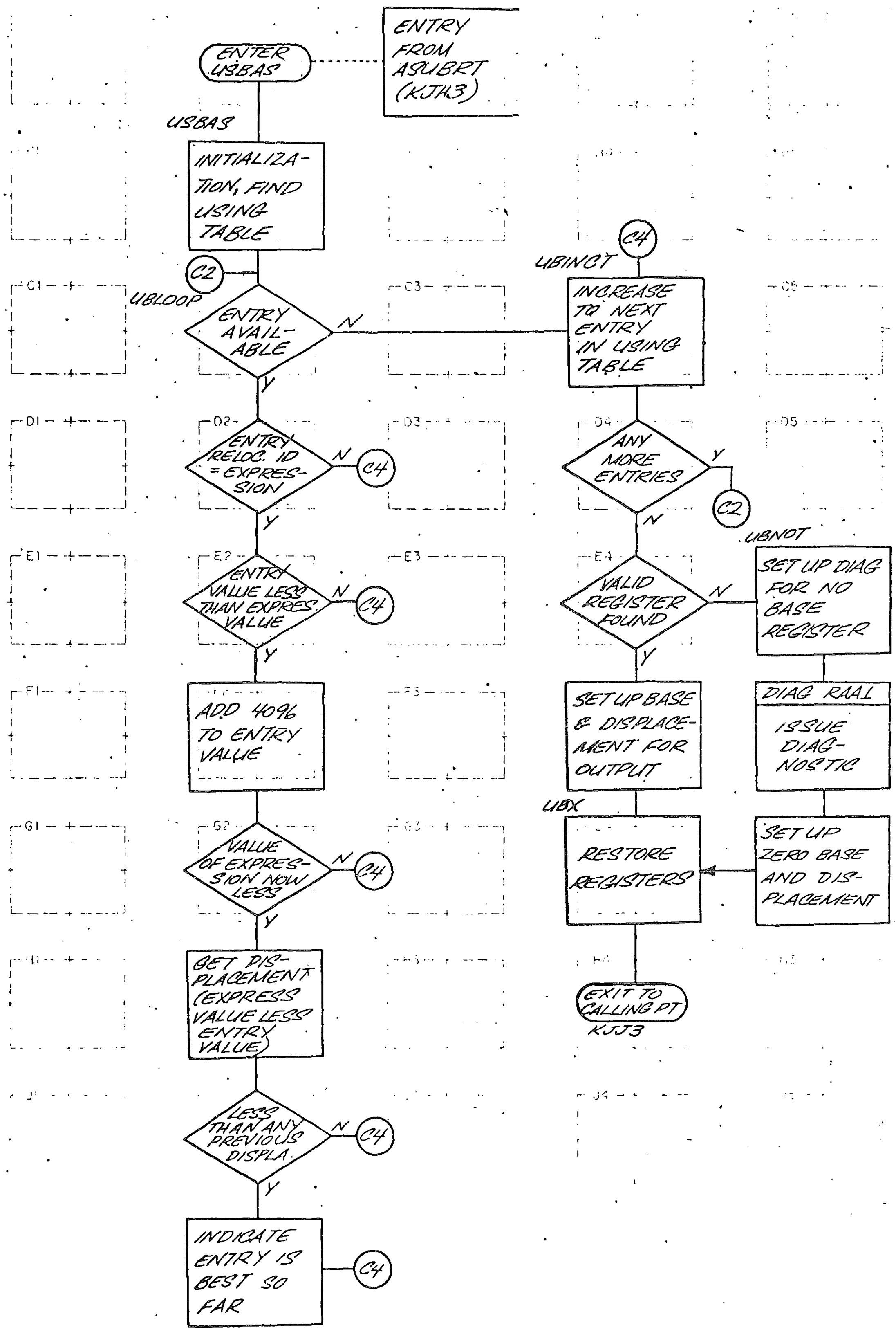
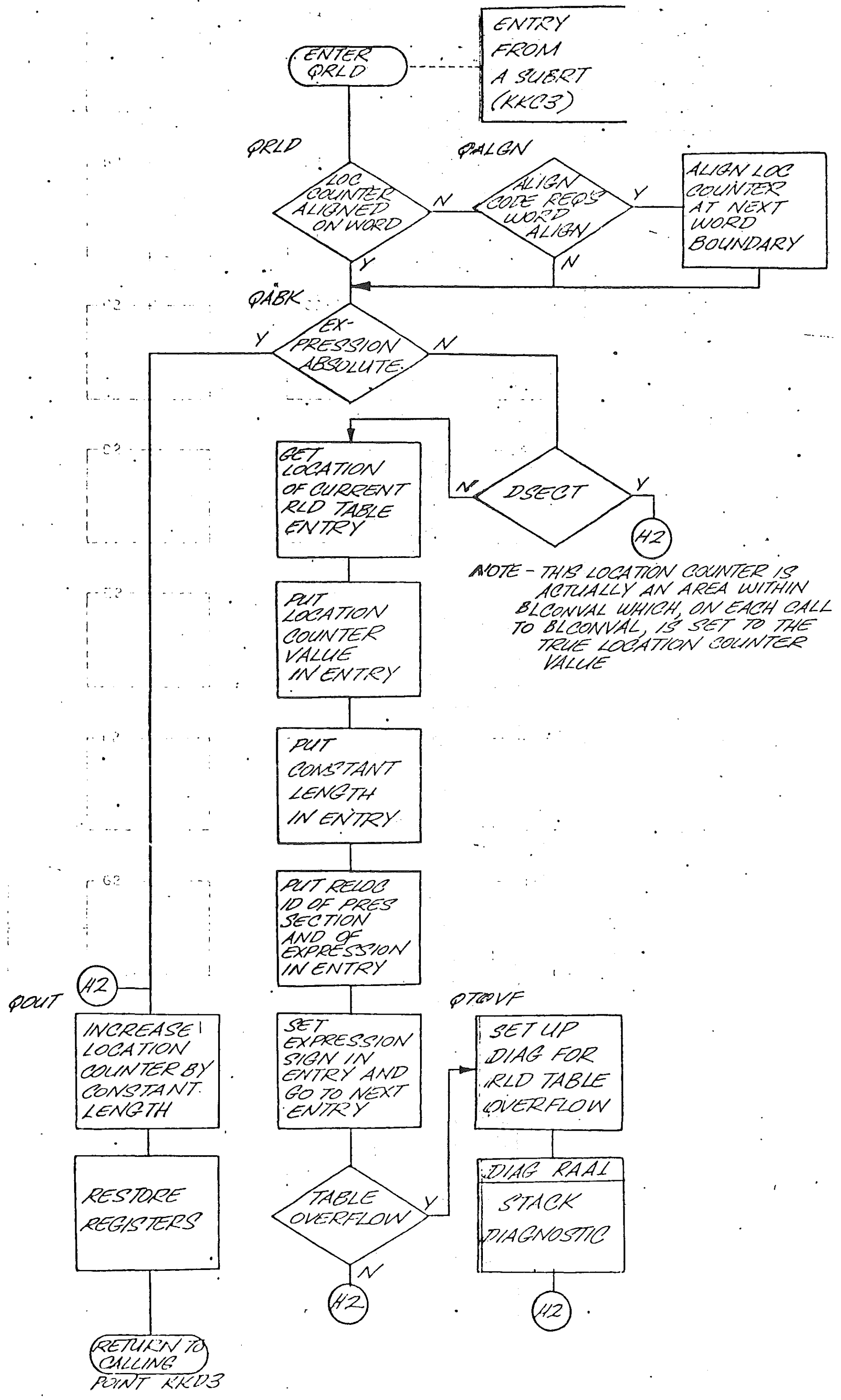
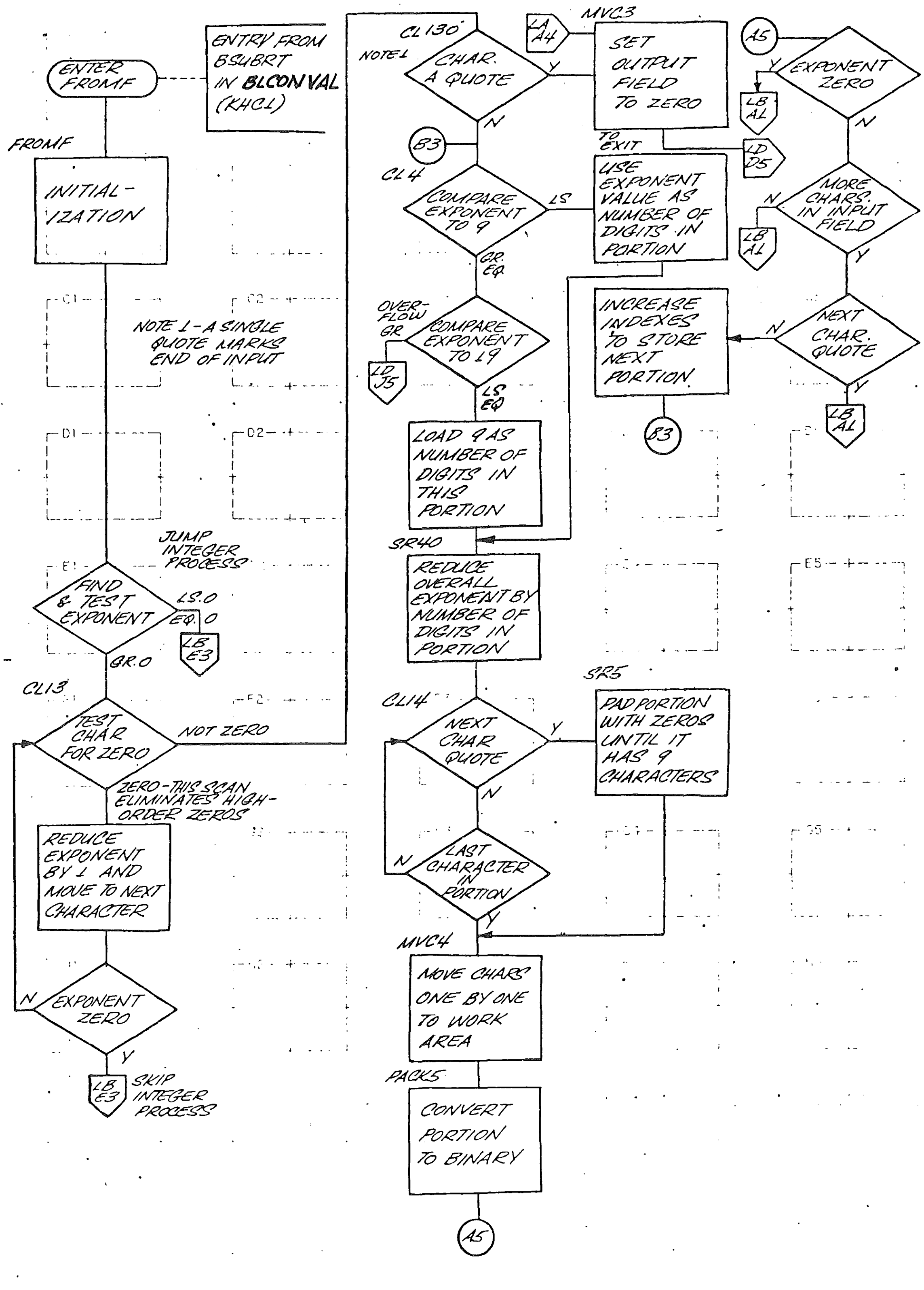

ENTRY FROM ASUBRT (KJK3)
ENTER USBAS
USBAS
INITIALIZATION, FIND USING TABLE
C2
UBLOOP
ENTRY AVAILABLE
N
Y
ENTRY RELOC. ID = EXPRESSION
N
C4
Y
ENTRY VALUE LESS THAN EXPRES. VALUE
N
C4
Y
ADD 4096 TO ENTRY VALUE
VALUE OF EXPRESSION NOW LESS
N
C4
Y
GET DISPLACEMENT (EXPRESS VALUE LESS ENTRY VALUE)
LESS THAN ANY PREVIOUS DISPLA.
N
C4
Y
INDICATE ENTRY IS BEST SO FAR
C4
C4
UBINCT
INCREASE TO NEXT ENTRY IN USING TABLE
ANY MORE ENTRIES
Y
C2
N
VALID REGISTER FOUND
N
Y
UBNOT
SET UP DIAG FOR NO BASE REGISTER
DIAG RAAL
ISSUE DIAGNOSTIC
SET UP BASE & DISPLACEMENT FOR OUTPUT
UBX
RESTORE REGISTERS
SET UP ZERO BASE AND DISPLACEMENT
EXIT TO CALLING PT
KJJ3

CHART KL. USBAS SUBROUTINE



ENTER QRLD
ENTRY FROM A SUBRT (KKC3)
QRLD
LOC COUNTER ALIGNED ON WORD
N
Y
QALGN
ALIGN CODE REQS WORD ALIGN
Y
N
ALIGN LOC COUNTER AT NEXT WORD BOUNDARY
QABK
EXPRESSION ABSOLUTE
Y
N
DSECT
N
Y
H2
GET LOCATION OF CURRENT RLD TABLE ENTRY
NOTE - THIS LOCATION COUNTER IS ACTUALLY AN AREA WITHIN BLCONVAL WHICH, ON EACH CALL TO BLCONVAL, IS SET TO THE TRUE LOCATION COUNTER VALUE
PUT LOCATION COUNTER VALUE IN ENTRY
PUT CONSTANT LENGTH IN ENTRY
PUT RELOC ID OF PRES SECTION AND OF EXPRESSION IN ENTRY
QOUT
H2
INCREASE LOCATION COUNTER BY CONSTANT LENGTH
SET EXPRESSION SIGN IN ENTRY AND GO TO NEXT ENTRY
QTOVF
SET UP DIAG FOR RLD TABLE OVERFLOW
RESTORE REGISTERS
TABLE OVERFLOW
Y
N
H2
DIAG RAAL
STACK DIAGNOSTIC
H2
RETURN TO CALLING POINT KKD3

CHART KM. QRLD SUBROUTINE



ENTER FROMF
ENTRY FROM BSUBRT IN BLCONVAL (KHCL)
FROMF
INITIAL-IZATION
NOTE 1 - A SINGLE QUOTE MARKS END OF INPUT
JUMP INTEGER PROCESS
FIND & TEST EXPONENT
LS.0
EQ. 0
LB E3
GR.0
CL13
TEST CHAR FOR ZERO
NOT ZERO
ZERO-THIS SCAN ELIMINATES HIGH-ORDER ZEROS
REDUCE EXPONENT BY 1 AND MOVE TO NEXT CHARACTER
N
EXPONENT ZERO
Y
LB E3
SKIP INTEGER PROCESS
CL130
NOTE 1
CHAR. A QUOTE
Y
N
B3
CL4
COMPARE EXPONENT TO 9
LS
GR. EQ
OVER-FLOW GR
COMPARE EXPONENT TO 19
LD J5
LS EQ
LOAD 9 AS NUMBER OF DIGITS IN THIS PORTION
SR40
REDUCE OVERALL EXPONENT BY NUMBER OF DIGITS IN PORTION
CL14
NEXT CHAR QUOTE
Y
N
LAST CHARACTER IN PORTION
N
Y
MVC4
MOVE CHARS ONE BY ONE TO WORK AREA
PACKS
CONVERT PORTION TO BINARY
A5
LA A4
MVC3
SET OUTPUT FIELD TO ZERO
TO EXIT
LD D5
USE EXPONENT VALUE AS NUMBER OF DIGITS IN PORTION
INCREASE INDEXES TO STORE NEXT PORTION
B3
SR5
PAD PORTION WITH ZEROS UNTIL IT HAS 9 CHARACTERS
A5
Y
EXPONENT ZERO
LB A1
N
MORE CHARS IN INPUT FIELD
N
LB A1
Y
NEXT CHAR. QUOTE
N
Y
LB A1

CHART LA. FROMF ROUTINE (SHEET 1 OF 5)

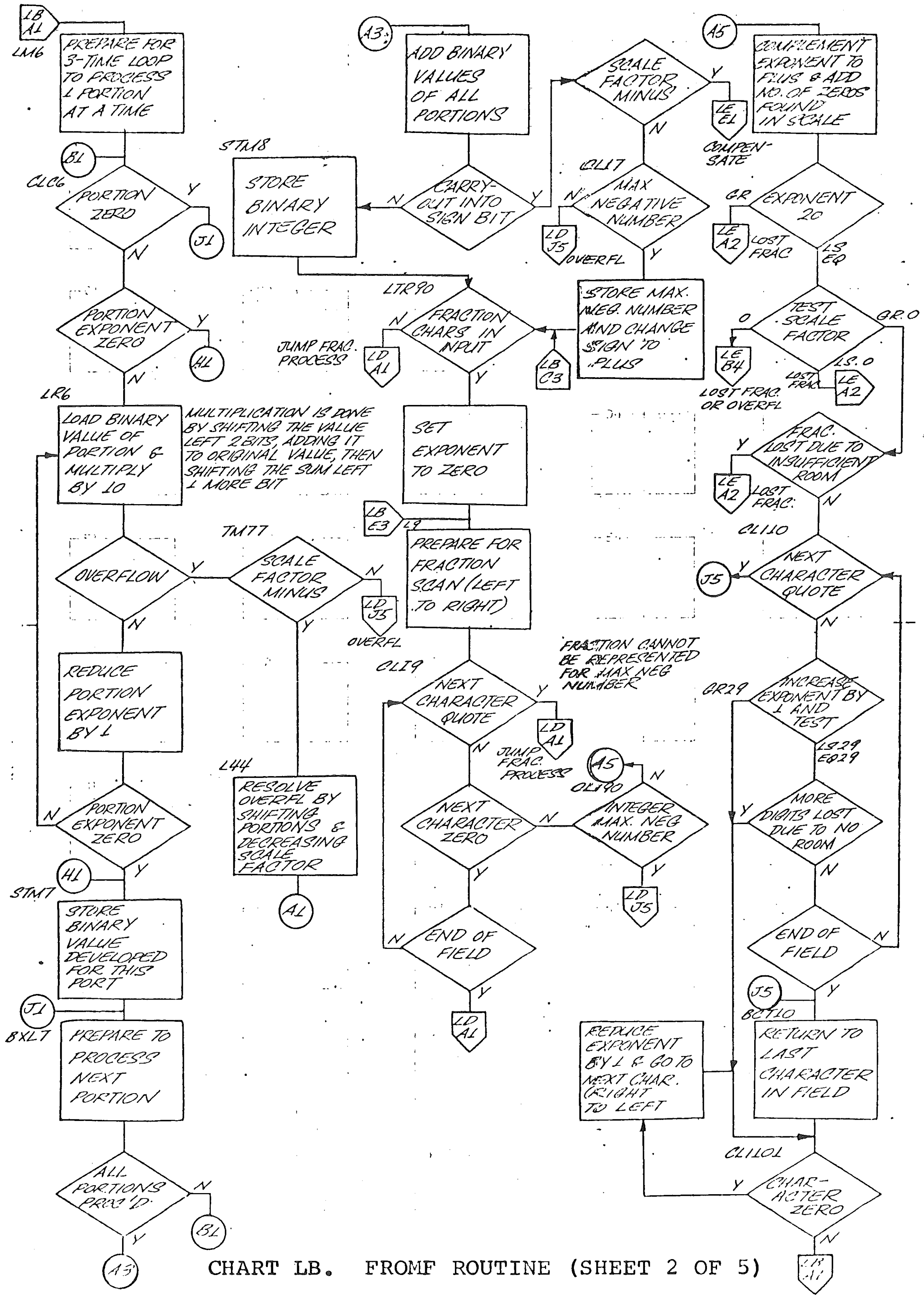

LB A1
LM6
PREPARE FOR 3-TIME LOOP TO PROCESS 1 PORTION AT A TIME
B1
CLC6
PORTION ZERO
J1
PORTION EXPONENT ZERO
H1
LR6
LOAD BINARY VALUE OF PORTION & MULTIPLY BY 10
MULTIPLICATION IS DONE BY SHIFTING THE VALUE LEFT 2 BITS, ADDING IT TO ORIGINAL VALUE, THEN SHIFTING THE SUM LEFT 1 MORE BIT
OVERFLOW
TM77
SCALE FACTOR MINUS
LD J5
OVERFL
REDUCE PORTION EXPONENT BY 1
L44
RESOLVE OVERFL BY SHIFTING PORTIONS & DECREASING SCALE FACTOR
A1
PORTION EXPONENT ZERO
STM7
STORE BINARY VALUE DEVELOPED FOR THIS PORT
BXL7
PREPARE TO PROCESS NEXT PORTION
ALL PORTIONS PROC'D
B1
A3
ADD BINARY VALUES OF ALL PORTIONS
STM8
STORE BINARY INTEGER
CARRY-OUT INTO SIGN BIT
SCALE FACTOR MINUS
LE E1
COMPEN-SATE
CLL7
MAX NEGATIVE NUMBER
LD J5
OVERFL
STORE MAX. NEG. NUMBER AND CHANGE SIGN TO PLUS
LTR90
FRACTION CHARS IN INPUT
JUMP FRAC. PROCESS
LD A1
LB C3
SET EXPONENT TO ZERO
LB E3
L9
PREPARE FOR FRACTION SCAN (LEFT TO RIGHT)
FRACTION CANNOT BE REPRESENTED FOR MAX NEG NUMBER
CLI9
NEXT CHARACTER QUOTE
LD A1
JUMP FRAC. PROCESS
A5
CLI90
NEXT CHARACTER ZERO
INTEGER MAX. NEG NUMBER
LD J5
END OF FIELD
LD A1
A5
COMPLEMENT EXPONENT TO PLUS & ADD NO. OF ZEROS FOUND IN SCALE
GR
EXPONENT 20
LE A2
LOST FRAC
LS EQ
TEST SCALE FACTOR
0
GR.0
LE B4
LOST FRAC. OR OVERFL
LS.0
LOST FRAC
LE A2
FRAC. LOST DUE TO INSUFFICIENT ROOM
LE A2
LOST FRAC.
CL110
NEXT CHARACTER QUOTE
J5
GR29
INCREASE EXPONENT BY 1 AND TEST
LS29 EQ29
MORE DIGITS LOST DUE TO NO ROOM
END OF FIELD
J5
BCT10
RETURN TO LAST CHARACTER IN FIELD
REDUCE EXPONENT BY 1 & GO TO NEXT CHAR. (RIGHT TO LEFT
CL1101
CHAR-ACTER ZERO
LB A1
Y
N

CHART LB. FROMF ROUTINE (SHEET 2 OF 5)

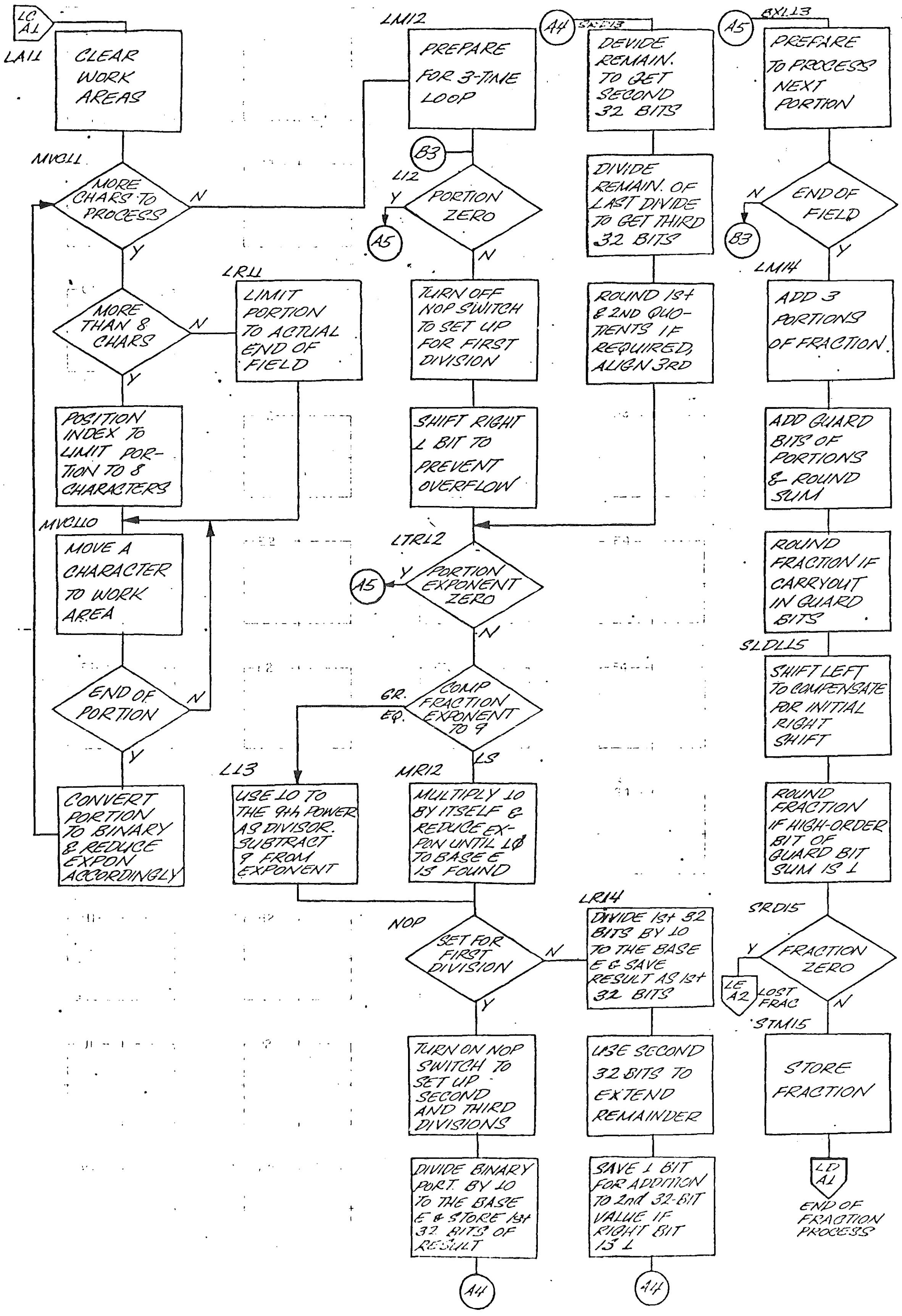
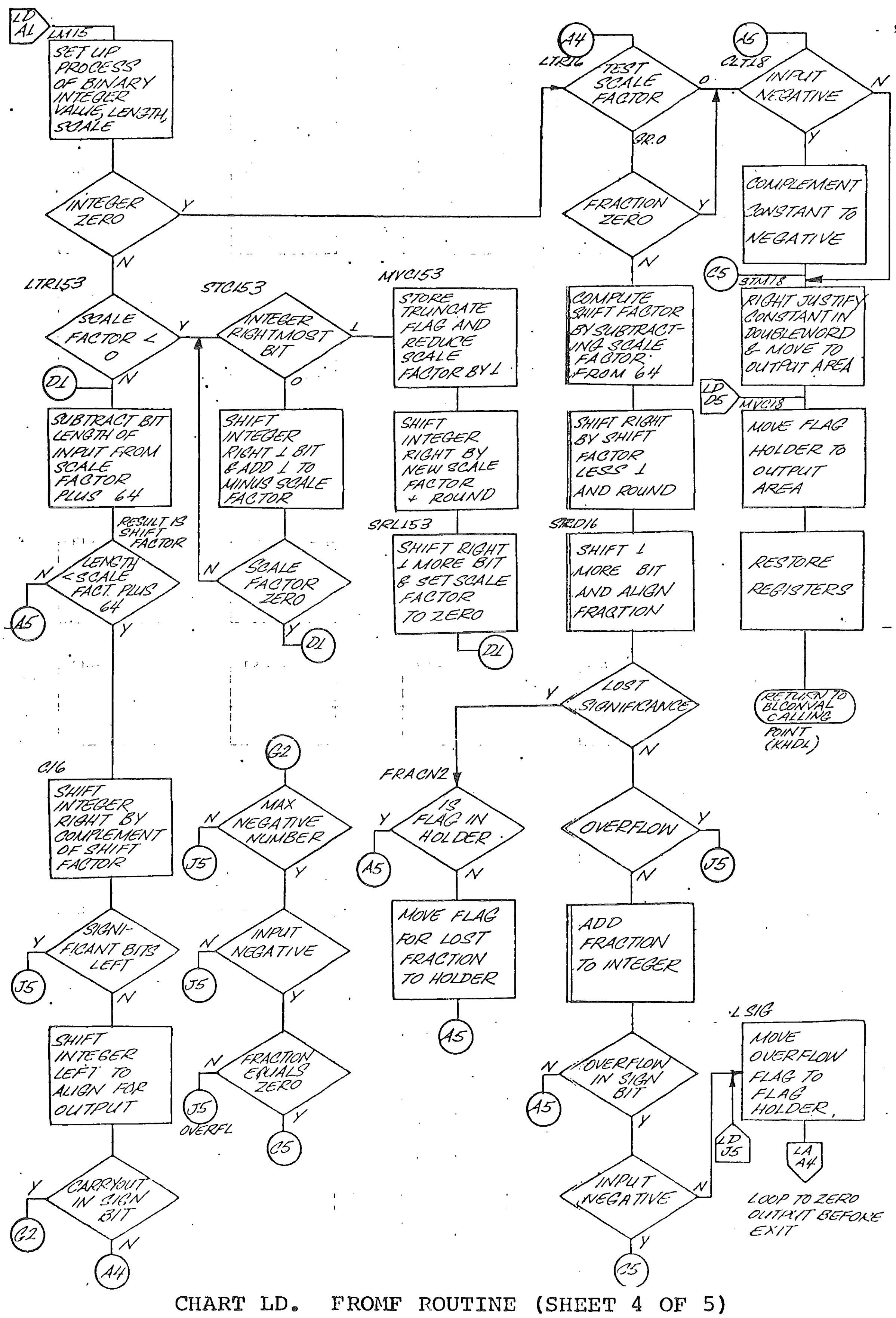

LC A1
LA11
CLEAR WORK AREAS
MVC11
MORE CHARS TO PROCESS
N
Y
MORE THAN 8 CHARS
LR11
LIMIT PORTION TO ACTUAL END OF FIELD
POSITION INDEX TO LIMIT PORTION TO 8 CHARACTERS
MVC110
MOVE A CHARACTER TO WORK AREA
END OF PORTION
CONVERT PORTION TO BINARY & REDUCE EXPON ACCORDINGLY
LM12
PREPARE FOR 3-TIME LOOP
B3
L12
PORTION ZERO
A5
TURN OFF NOP SWITCH TO SET UP FOR FIRST DIVISION
SHIFT RIGHT 1 BIT TO PREVENT OVERFLOW
LTR12
PORTION EXPONENT ZERO
COMP FRACTION EXPONENT TO 9
GR. EQ.
LS
L13
USE 10 TO THE 9th POWER AS DIVISOR. SUBTRACT 9 FROM EXPONENT
MR12
MULTIPLY 10 BY ITSELF & REDUCE EXPON UNTIL 10 TO BASE E IS FOUND
NOP
SET FOR FIRST DIVISION
TURN ON NOP SWITCH TO SET UP SECOND AND THIRD DIVISIONS
DIVIDE BINARY PORT. BY 10 TO THE BASE E & STORE 1st 32 BITS OF RESULT
A4
DEVIDE REMAIN. TO GET SECOND 32 BITS
DIVIDE REMAIN. OF LAST DIVIDE TO GET THIRD 32 BITS
ROUND 1st & 2ND QUOTIENTS IF REQUIRED, ALIGN 3RD
LR14
DIVIDE 1st 32 BITS BY 10 TO THE BASE E & SAVE RESULT AS 1st 32 BITS
USE SECOND 32 BITS TO EXTEND REMAINDER
SAVE 1 BIT FOR ADDITION TO 2nd 32-BIT VALUE IF RIGHT BIT IS 1
BXL13
PREPARE TO PROCESS NEXT PORTION
END OF FIELD
LM14
ADD 3 PORTIONS OF FRACTION
ADD GUARD BITS OF PORTIONS & ROUND SUM
ROUND FRACTION IF CARRYOUT IN GUARD BITS
SLDL15
SHIFT LEFT TO COMPENSATE FOR INITIAL RIGHT SHIFT
ROUND FRACTION IF HIGH-ORDER BIT OF GUARD BIT SUM IS 1
SRDI5
FRACTION ZERO
LE A2
LOST FRAC
STM15
STORE FRACTION
LD A1
END OF FRACTION PROCESS

CHART LC. FROMF ROUTINE (SHEET 3 OF 5)



LD A1
LM15
SET UP PROCESS OF BINARY INTEGER VALUE, LENGTH, SCALE
INTEGER ZERO
Y
N
LTR153
SCALE FACTOR < 0
Y
N
D1
SUBTRACT BIT LENGTH OF INPUT FROM SCALE FACTOR PLUS 64
RESULT IS SHIFT FACTOR
LENGTH < SCALE FACT. PLUS 64
N
A5
Y
C16
SHIFT INTEGER RIGHT BY COMPLEMENT OF SHIFT FACTOR
SIGNIFICANT BITS LEFT
Y
J5
N
SHIFT INTEGER LEFT TO ALIGN FOR OUTPUT
CARRYOUT IN SIGN BIT
Y
G2
N
A4
STC153
INTEGER RIGHTMOST BIT
1
0
SHIFT INTEGER RIGHT 1 BIT & ADD 1 TO MINUS SCALE FACTOR
SCALE FACTOR ZERO
N
Y
D1
G2
MAX NEGATIVE NUMBER
N
J5
Y
INPUT NEGATIVE
N
J5
Y
FRACTION EQUALS ZERO
N
J5
OVERFL
Y
C5
MVC153
STORE TRUNCATE FLAG AND REDUCE SCALE FACTOR BY 1
SHIFT INTEGER RIGHT BY NEW SCALE FACTOR + ROUND
SRL153
SHIFT RIGHT 1 MORE BIT & SET SCALE FACTOR TO ZERO
D1
FRACN2
IS FLAG IN HOLDER
Y
A5
N
MOVE FLAG FOR LOST FRACTION TO HOLDER
A5
A4
LTR16
TEST SCALE FACTOR
0
SR.0
FRACTION ZERO
Y
N
COMPUTE SHIFT FACTOR BY SUBTRACTING SCALE FACTOR FROM 64
SHIFT RIGHT BY SHIFT FACTOR LESS 1 AND ROUND
SRD16
SHIFT 1 MORE BIT AND ALIGN FRACTION
LOST SIGNIFICANCE
Y
N
OVERFLOW
Y
J5
N
ADD FRACTION TO INTEGER
OVERFLOW IN SIGN BIT
N
A5
Y
INPUT NEGATIVE
N
Y
C5
A5
CLT18
INPUT NEGATIVE
N
Y
COMPLEMENT CONSTANT TO NEGATIVE
C5
STM18
RIGHT JUSTIFY CONSTANT IN DOUBLEWORD & MOVE TO OUTPUT AREA
LD D5
MVC18
MOVE FLAG HOLDER TO OUTPUT AREA
RESTORE REGISTERS
RETURN TO BLCONVAL CALLING POINT (KHDL)
LSIG
MOVE OVERFLOW FLAG TO FLAG HOLDER.
LD J5
LA A4
LOOP TO ZERO OUTPUT BEFORE EXIT

CHART LD. FROMF ROUTINE (SHEET 4 OF 5)

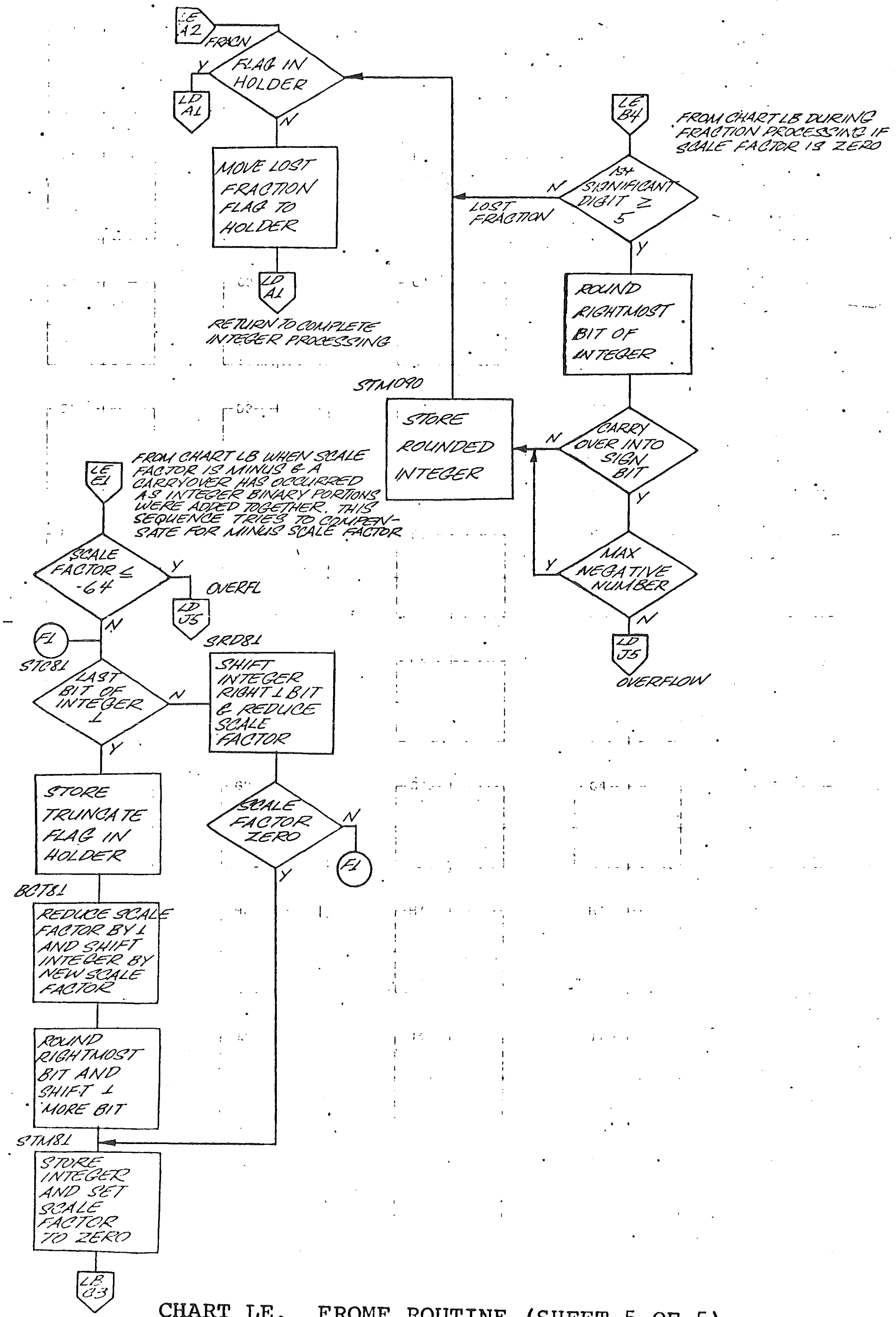

LE A2
FRACN
FLAG IN HOLDER
Y
LD A1
N
MOVE LOST FRACTION FLAG TO HOLDER
LD A1
RETURN TO COMPLETE INTEGER PROCESSING
LE B4
FROM CHART LB DURING FRACTION PROCESSING IF SCALE FACTOR IS ZERO
1st SIGNIFICANT DIGIT ≥ 5
N
LOST FRACTION
Y
ROUND RIGHTMOST BIT OF INTEGER
STM090
STORE ROUNDED INTEGER
N
CARRY OVER INTO SIGN BIT
Y
MAX NEGATIVE NUMBER
Y
N
LD J5
OVERFLOW
LE E1
FROM CHART LB WHEN SCALE FACTOR IS MINUS & A CARRYOVER HAS OCCURRED AS INTEGER BINARY PORTIONS WERE ADDED TOGETHER. THIS SEQUENCE TRIES TO COMPENSATE FOR MINUS SCALE FACTOR
SCALE FACTOR ≤ -64
Y
OVERFL
LD J5
N
F1
STC81
LAST BIT OF INTEGER 1
N
SRD81
SHIFT INTEGER RIGHT 1 BIT & REDUCE SCALE FACTOR
Y
STORE TRUNCATE FLAG IN HOLDER
SCALE FACTOR ZERO
N
F1
Y
BCT81
REDUCE SCALE FACTOR BY 1 AND SHIFT INTEGER BY NEW SCALE FACTOR
ROUND RIGHTMOST BIT AND SHIFT 1 MORE BIT
STM81
STORE INTEGER AND SET SCALE FACTOR TO ZERO
LB C3

CHART LE. FROMF ROUTINE (SHEET 5 OF 5)

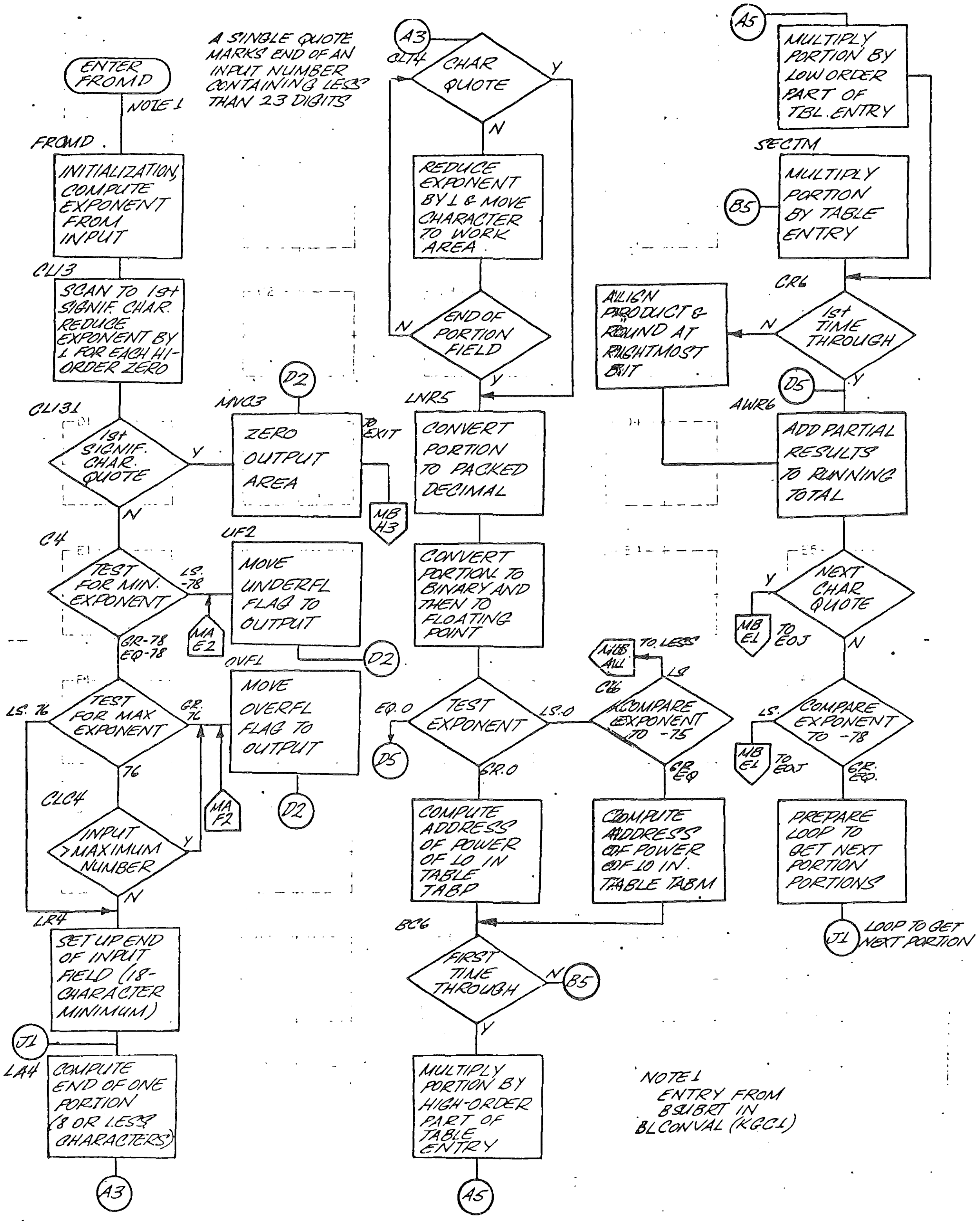

A SINGLE QUOTE MARKS END OF AN INPUT NUMBER CONTAINING LESS THAN 23 DIGITS
ENTER FROMD
NOTE 1
FROMD
INITIALIZATION, COMPUTE EXPONENT FROM INPUT
CL13
SCAN TO 1st SIGNIF CHAR. REDUCE EXPONENT BY 1 FOR EACH HI-ORDER ZERO
CL131
1st SIGNIF. CHAR. QUOTE
MVC3
ZERO OUTPUT AREA
TO EXIT
MB H3
D2
C4
TEST FOR MIN. EXPONENT
LS. -78
GR-78 EQ-78
UF2
MOVE UNDERFL FLAG TO OUTPUT
MA E2
OVFL
MOVE OVERFL FLAG TO OUTPUT
TEST FOR MAX EXPONENT
LS. 76
GR. 76
76
CLC4
INPUT >MAXIMUM NUMBER
MA F2
LR4
SET UP END OF INPUT FIELD (18-CHARACTER MINIMUM)
J1
LA4
COMPUTE END OF ONE PORTION (8 OR LESS CHARACTERS)
A3
CLT4
CHAR QUOTE
REDUCE EXPONENT BY 1 & MOVE CHARACTER TO WORK AREA
END OF PORTION FIELD
LNR5
CONVERT PORTION TO PACKED DECIMAL
CONVERT PORTION TO BINARY AND THEN TO FLOATING POINT
TEST EXPONENT
EQ.0
LS.0
GR.0
D5
COMPUTE ADDRESS OF POWER OF 10 IN TABLE TABP
BC6
FIRST TIME THROUGH
B5
MULTIPLY PORTION BY HIGH-ORDER PART OF TABLE ENTRY
A5
MB A11
TO LESS
LS
C6
COMPARE EXPONENT TO -75
GR EQ
COMPUTE ADDRESS OF POWER OF 10 IN TABLE TABM
MULTIPLY PORTION BY LOW ORDER PART OF TBL. ENTRY
SECTM
MULTIPLY PORTION BY TABLE ENTRY
CR6
1st TIME THROUGH
ALIGN PRODUCT & ROUND AT RIGHTMOST BIT
AWR6
ADD PARTIAL RESULTS TO RUNNING TOTAL
NEXT CHAR QUOTE
MB E1
TO EOJ
COMPARE EXPONENT TO -78
LS.
GR. EQ.
PREPARE LOOP TO GET NEXT PORTION PORTIONS
LOOP TO GET NEXT PORTION
Y
N
NOTE 1 ENTRY FROM BSUBRT IN BLCONVAL (KGC1)

CHART MA. FROMD ROUTINE (SHEET 1 OF 2)

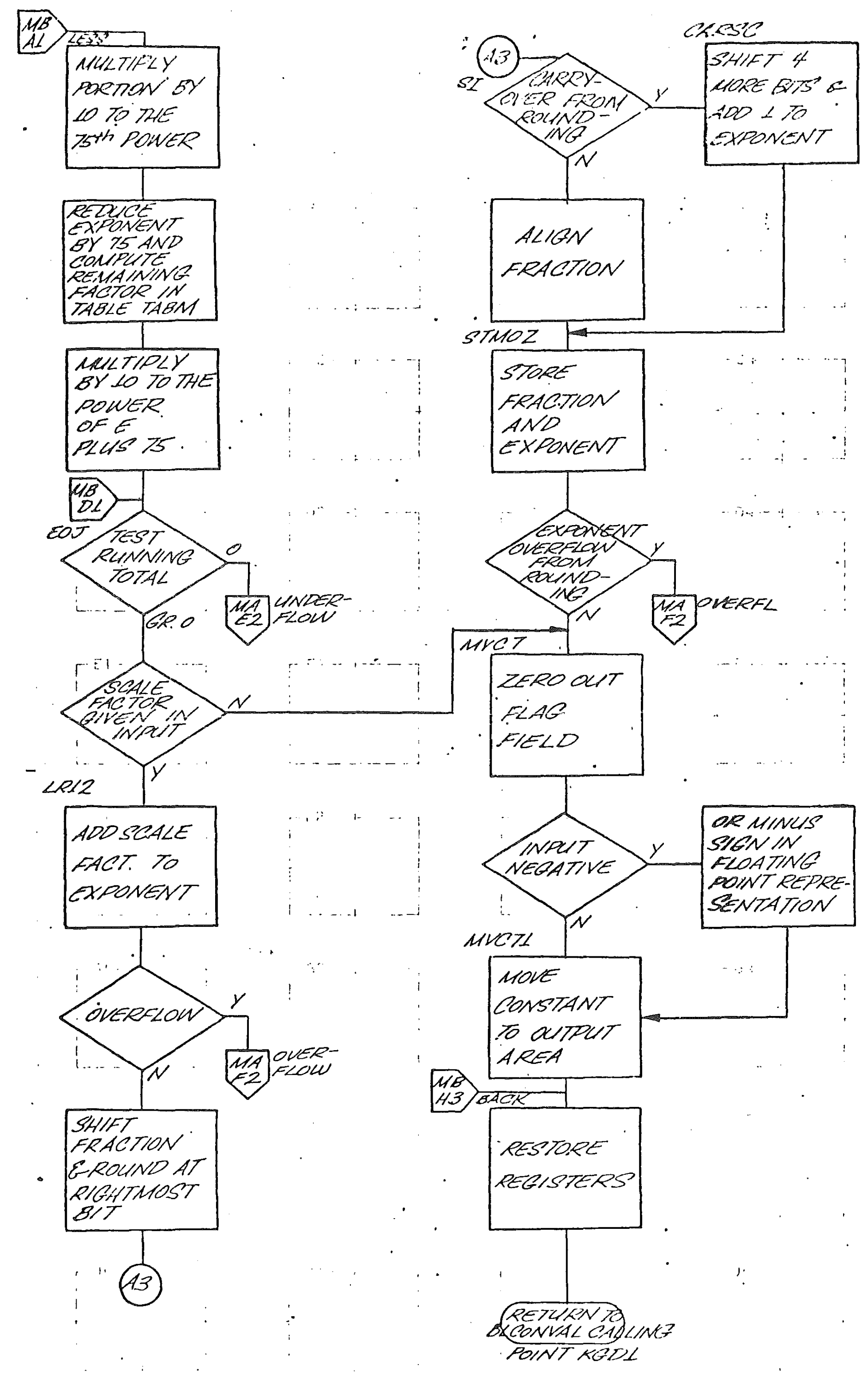

MB A1
LESS
MULTIPLY PORTION BY 10 TO THE 75th POWER
REDUCE EXPONENT BY 75 AND COMPUTE REMAINING FACTOR IN TABLE TABM
MULTIPLY BY 10 TO THE POWER OF E PLUS 75
MB D1
EOJ
TEST RUNNING TOTAL
0
MA E2
UNDER-FLOW
GR. 0
SCALE FACTOR GIVEN IN INPUT
N
Y
LR12
ADD SCALE FACT. TO EXPONENT
OVERFLOW
Y
MA F2
OVER-FLOW
N
SHIFT FRACTION & ROUND AT RIGHTMOST BIT
A3
A3
S1
CARRY-OVER FROM ROUND-ING
Y
CKRSC
SHIFT 4 MORE BITS & ADD 1 TO EXPONENT
N
ALIGN FRACTION
STMOZ
STORE FRACTION AND EXPONENT
EXPONENT OVERFLOW FROM ROUND-ING
Y
MA F2
OVERFL
N
MVC7
ZERO OUT FLAG FIELD
INPUT NEGATIVE
Y
OR MINUS SIGN IN FLOATING POINT REPRE-SENTATION
N
MVC71
MOVE CONSTANT TO OUTPUT AREA
MB H3
BACK
RESTORE REGISTERS
RETURN TO DECONVAL CALLING POINT KGD1

CHART MB. FROMD ROUTINE (SHEET 2 OF 2)

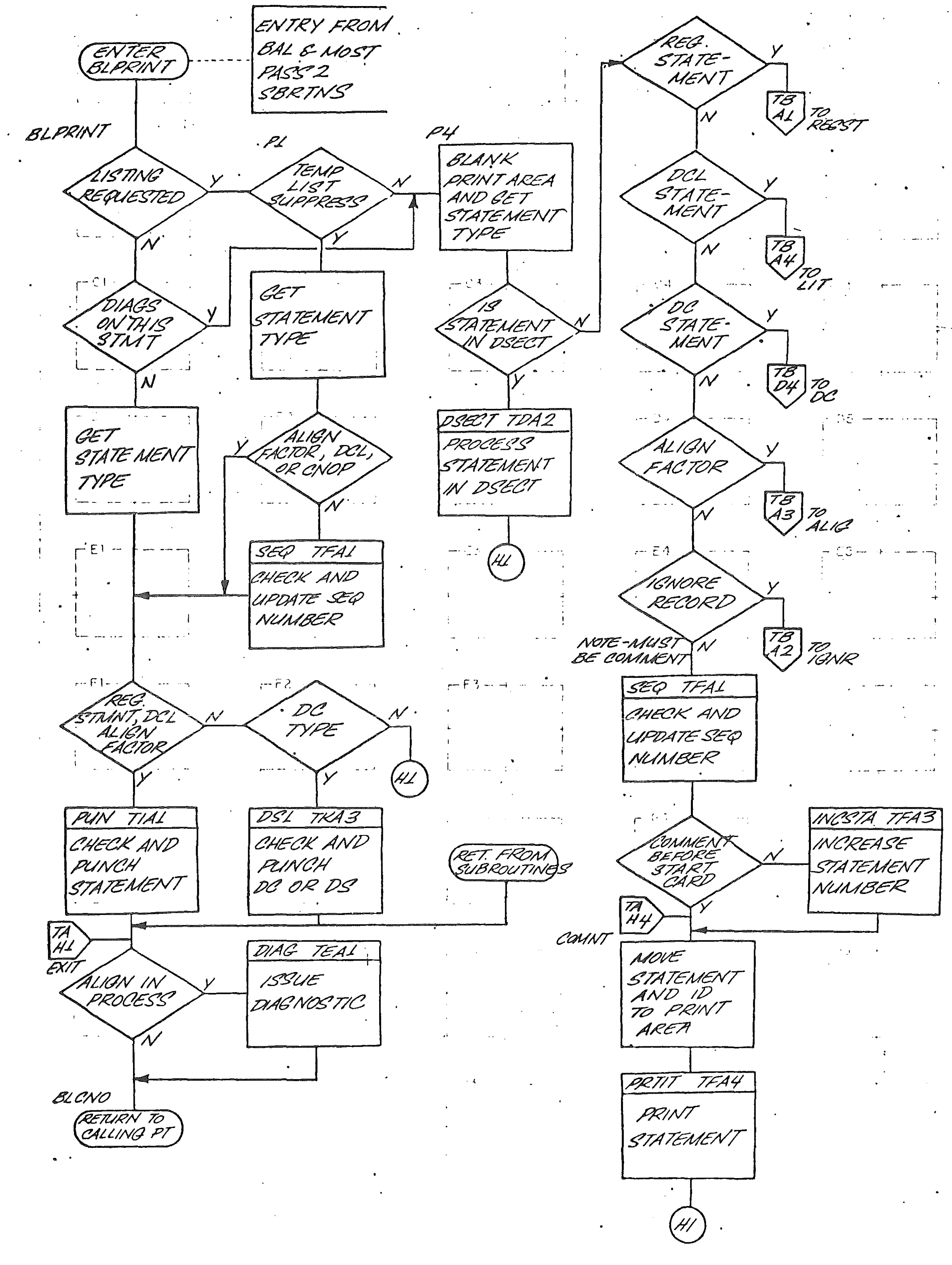


CHART TA. BLPRINT MAINLINE (SHEET 1 OF 3)

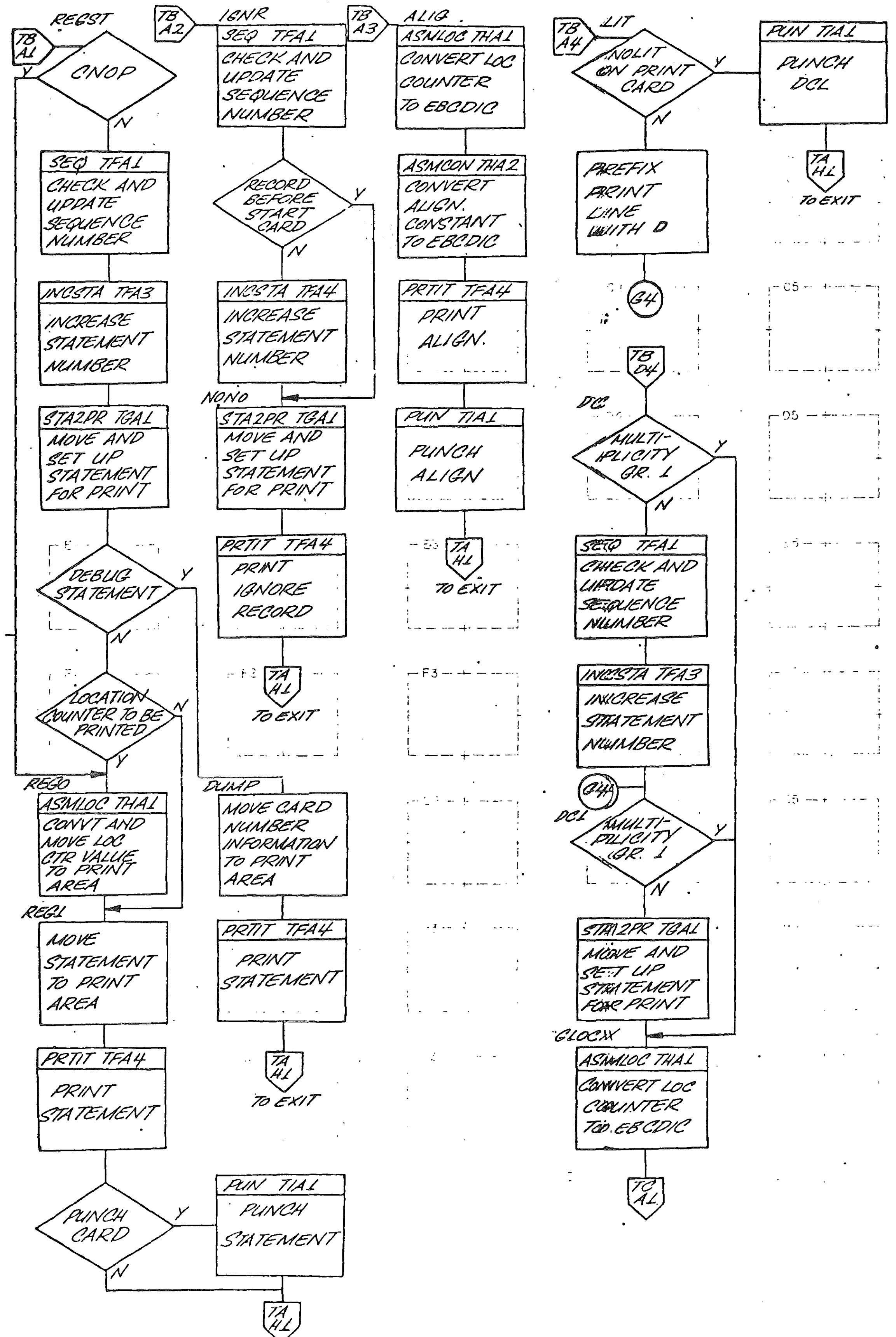


CHART TB. BLPRINT MAINLINE (SHEET 2 OF 3)

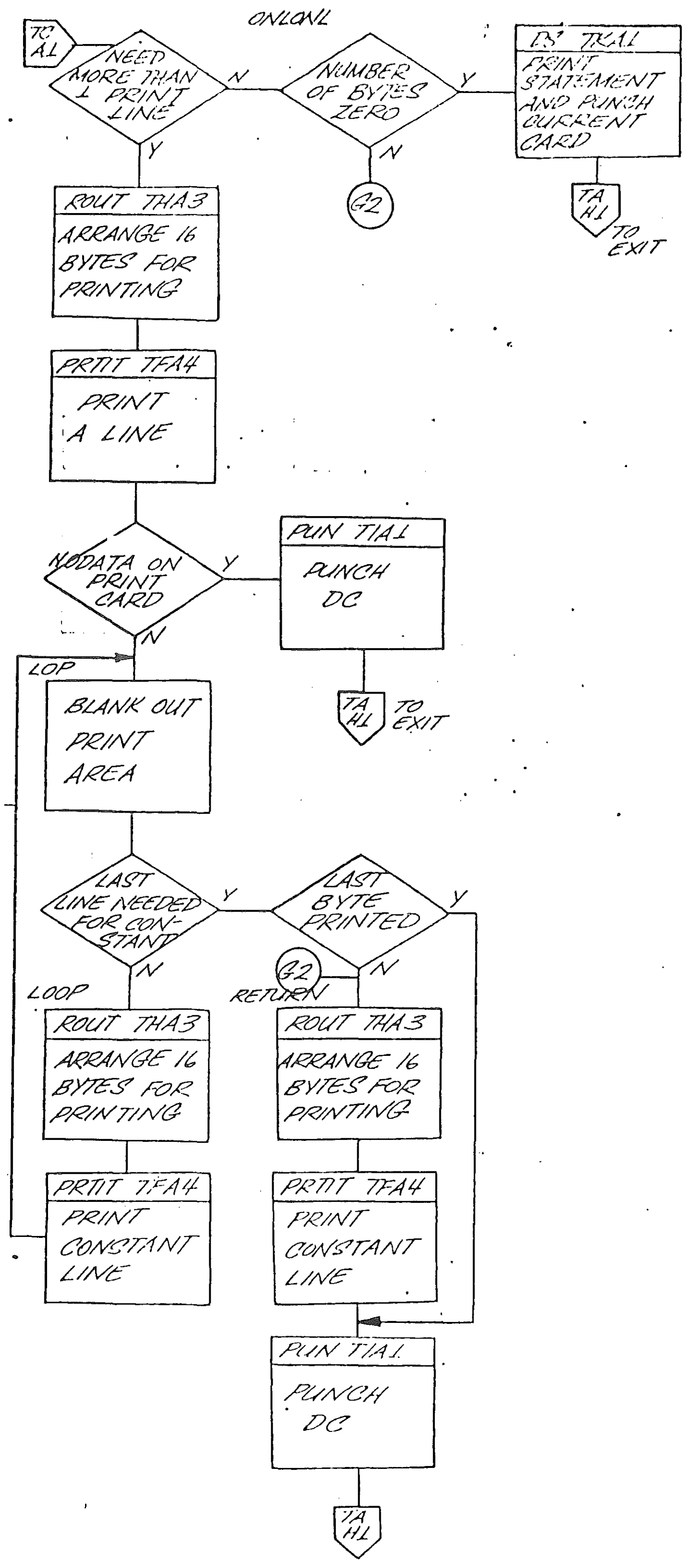


CHART TC. BLPRINT MAINLINE (SHEET 3 OF 3)

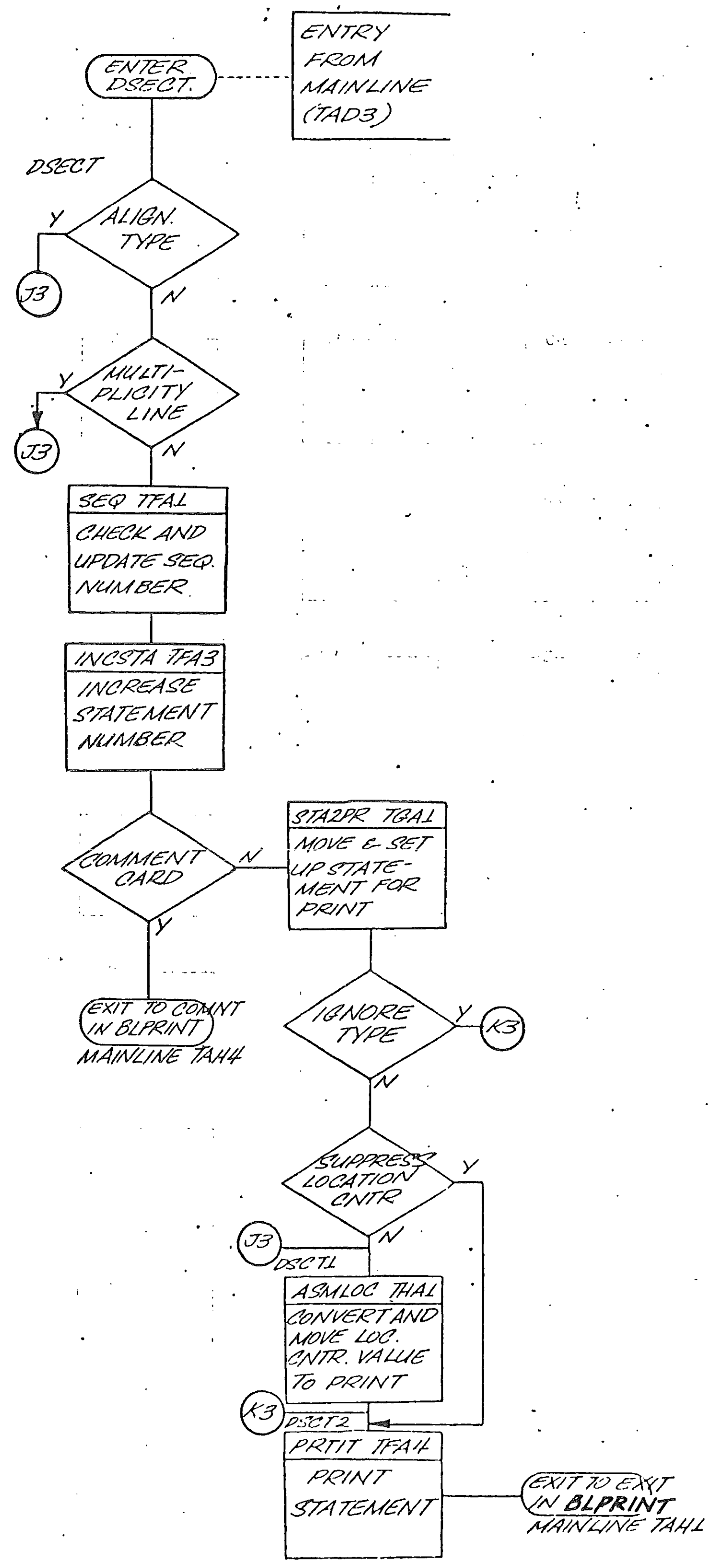
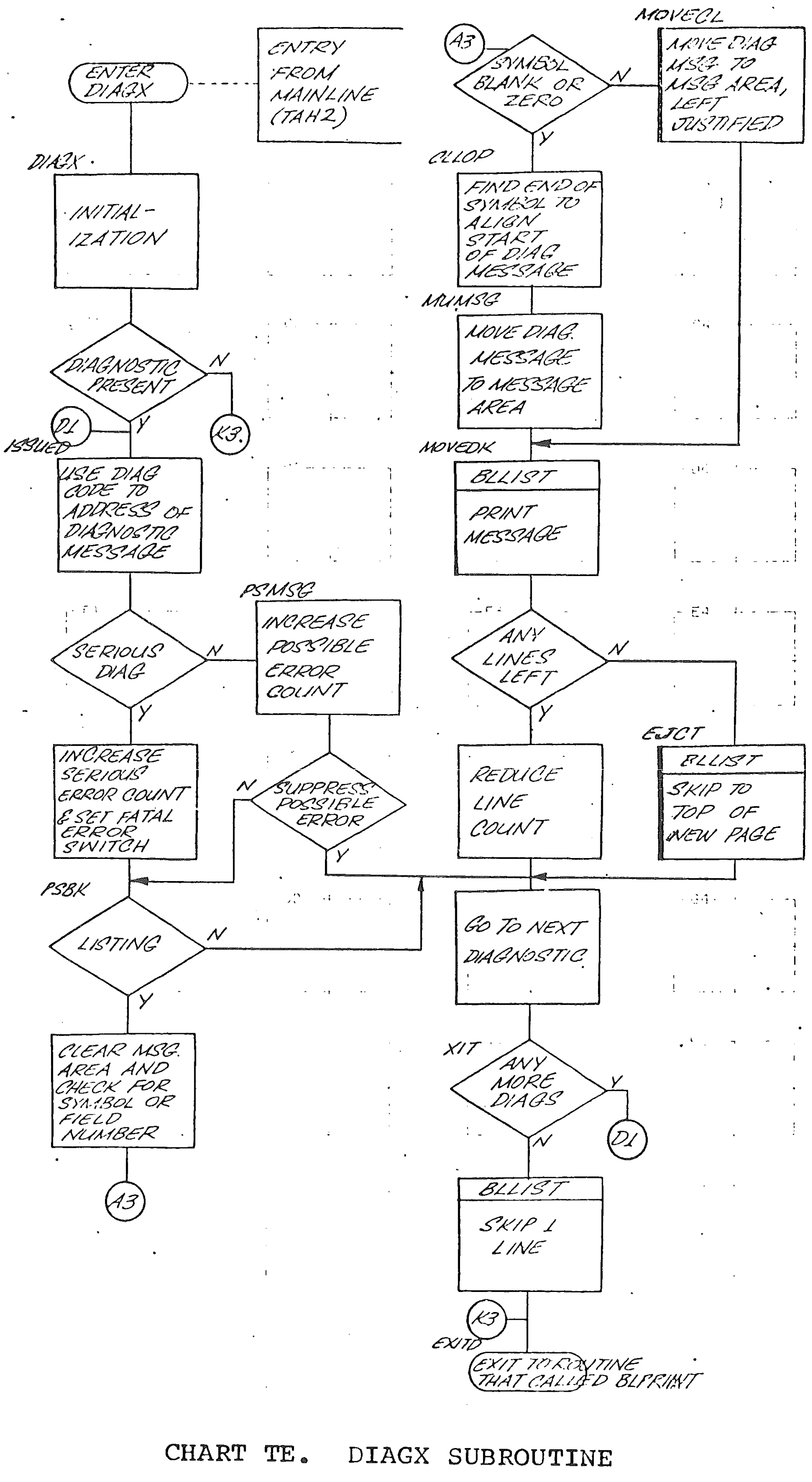
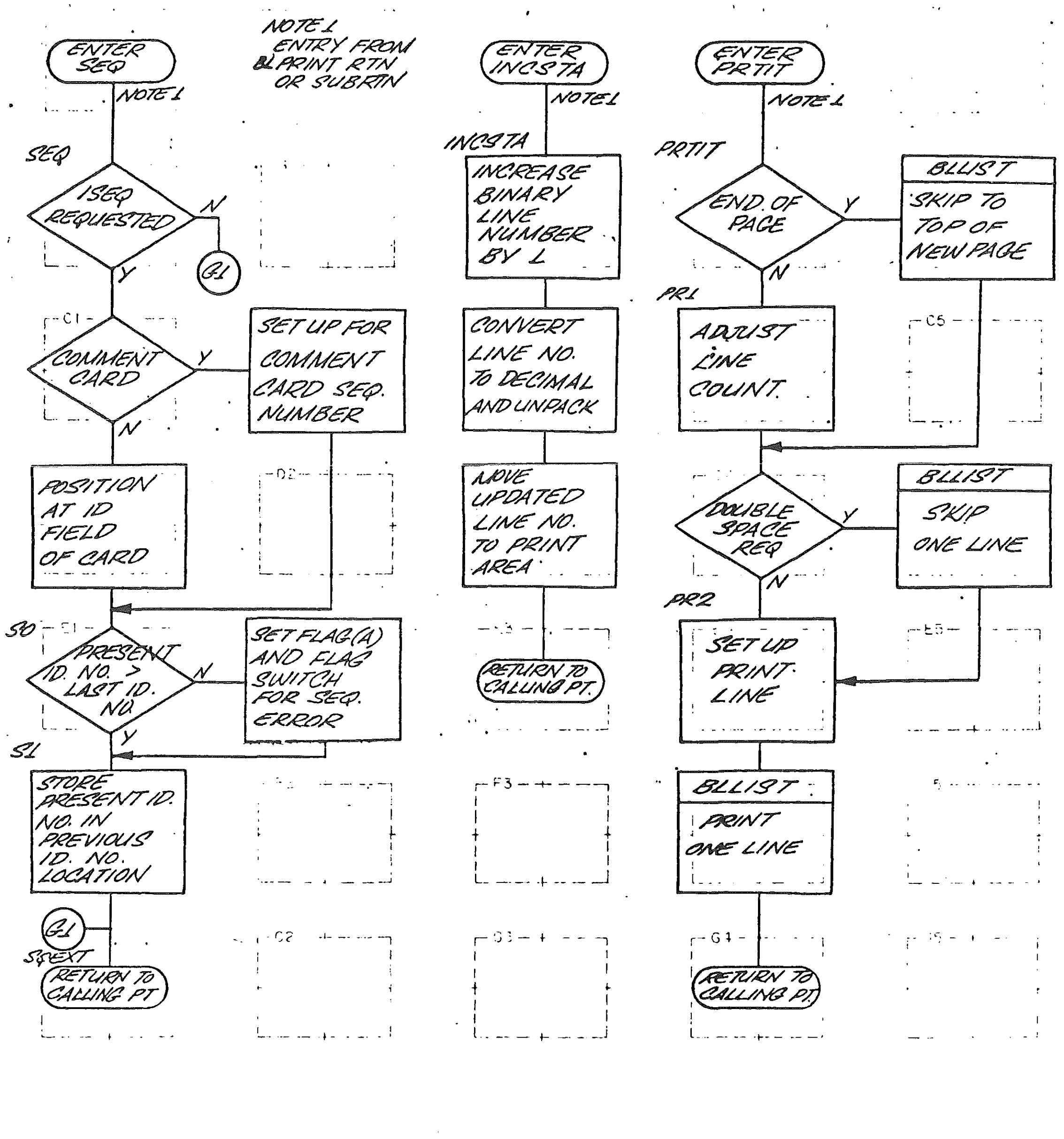


CHART TD. DSECT SUBROUTINE



ENTER DIAGX
ENTRY FROM MAINLINE (TAH2)
DIAGX
INITIAL-IZATION
DIAGNOSTIC PRESENT
N
Y
K3
D1
ISSUED
USE DIAG CODE TO ADDRESS OF DIAGNOSTIC MESSAGE
SERIOUS DIAG
N
Y
PSMSG
INCREASE POSSIBLE ERROR COUNT
INCREASE SERIOUS ERROR COUNT & SET FATAL ERROR SWITCH
SUPPRESS POSSIBLE ERROR
N
Y
PSBK
LISTING
N
Y
CLEAR MSG. AREA AND CHECK FOR SYMBOL OR FIELD NUMBER
A3
A3
SYMBOL BLANK OR ZERO
N
Y
MOVECL
MOVE DIAG MSG TO MSG AREA, LEFT JUSTIFIED
CLLOP
FIND END OF SYMBOL TO ALIGN START OF DIAG MESSAGE
MVMSG
MOVE DIAG. MESSAGE TO MESSAGE AREA
MOVEDK
BLLIST
PRINT MESSAGE
ANY LINES LEFT
N
Y
EJCT
BLLIST
SKIP TO TOP OF NEW PAGE
REDUCE LINE COUNT
GO TO NEXT DIAGNOSTIC
XIT
ANY MORE DIAGS
Y
N
D1
BLLIST
SKIP 1 LINE
K3
EXITD
EXIT TO ROUTINE THAT CALLED BLPRINT

CHART TE. DIAGX SUBROUTINE



NOTE 1 ENTRY FROM BLPRINT RTN OR SUBRTN
ENTER SEQ
NOTE 1
SEQ
ISEQ REQUESTED
N
G1
Y
C1
COMMENT CARD
Y
SET UP FOR COMMENT CARD SEQ. NUMBER
N
POSITION AT ID FIELD OF CARD
S0
PRESENT ID. NO. > LAST ID. NO.
N
SET FLAG(A) AND FLAG SWITCH FOR SEQ. ERROR
Y
S1
STORE PRESENT ID. NO. IN PREVIOUS ID. NO. LOCATION
G1
SQEXT
RETURN TO CALLING PT
ENTER INCSTA
NOTE 1
INCSTA
INCREASE BINARY LINE NUMBER BY 1
CONVERT LINE NO. TO DECIMAL AND UNPACK
MOVE UPDATED LINE NO. TO PRINT AREA
RETURN TO CALLING PT.
ENTER PRTIT
NOTE 1
PRTIT
END OF PAGE
Y
BLLIST
SKIP TO TOP OF NEW PAGE
N
PR1
ADJUST LINE COUNT
DOUBLE SPACE REQ
Y
BLLIST
SKIP ONE LINE
N
PR2
SET UP PRINT LINE
BLLIST
PRINT ONE LINE
RETURN TO CALLING PT.

CHART TF. SEQ, INCSTA, PRTIT SUBROUTINES

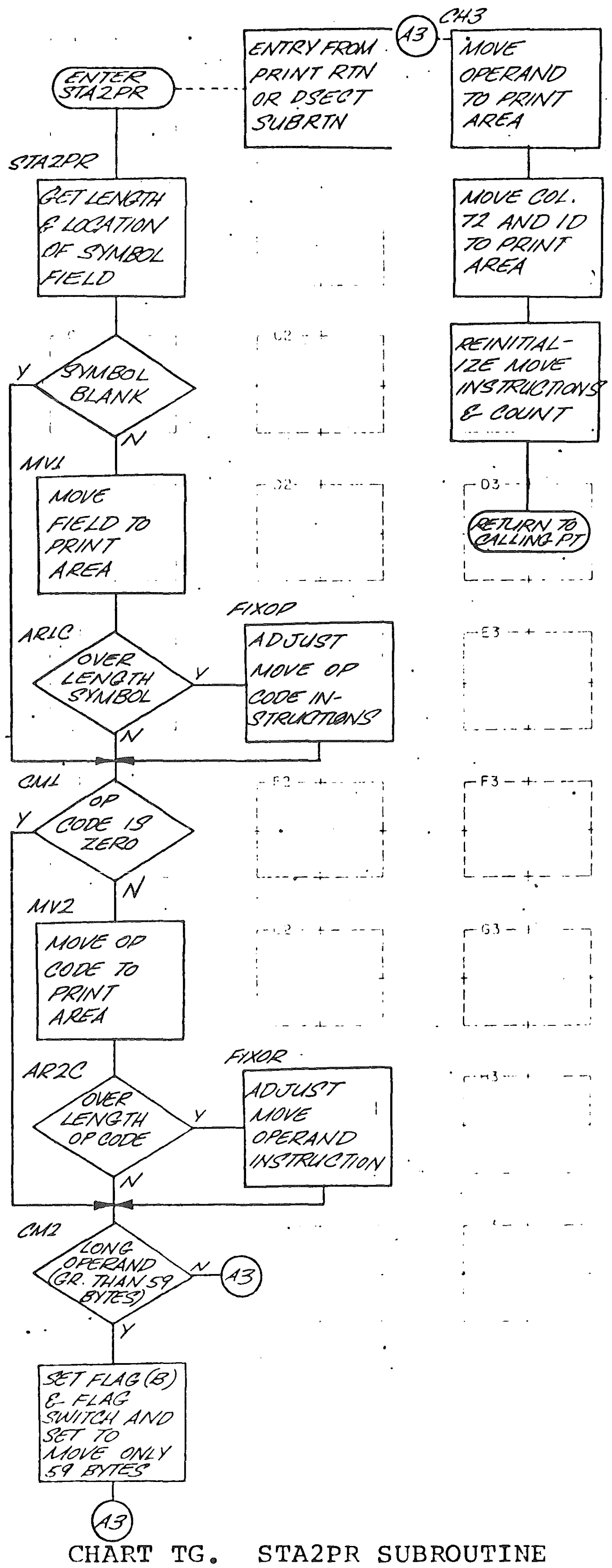
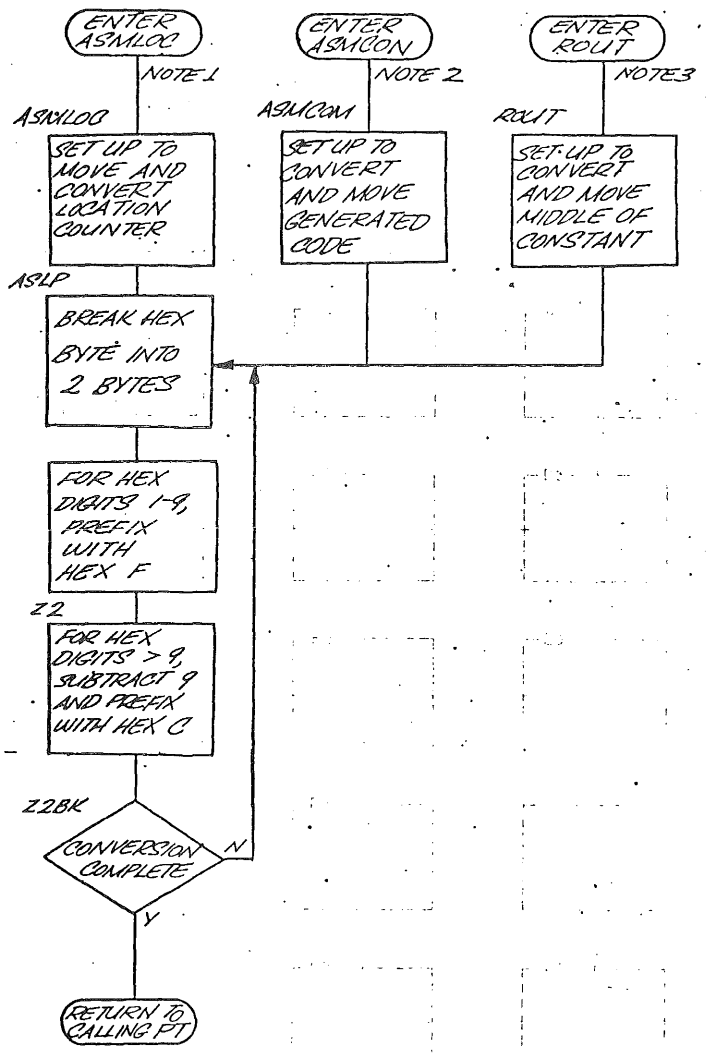

ENTER STA2PR
ENTRY FROM PRINT RTN OR DSECT SUBRTN
STA2PR
GET LENGTH & LOCATION OF SYMBOL FIELD
SYMBOL BLANK
Y
N
MV1
MOVE FIELD TO PRINT AREA
AR1C
OVER LENGTH SYMBOL
Y
N
FIXOP
ADJUST MOVE OP CODE INSTRUCTIONS
CM1
OP CODE IS ZERO
Y
N
MV2
MOVE OP CODE TO PRINT AREA
AR2C
OVER LENGTH OP CODE
Y
N
FIXOR
ADJUST MOVE OPERAND INSTRUCTION
CM2
LONG OPERAND (GR. THAN 59 BYTES)
N
A3
Y
SET FLAG (B) & FLAG SWITCH AND SET TO MOVE ONLY 59 BYTES
A3
A3
CH3
MOVE OPERAND TO PRINT AREA
MOVE COL. 72 AND ID TO PRINT AREA
REINITIALIZE MOVE INSTRUCTIONS & COUNT
D3
RETURN TO CALLING PT
E3
F3
G3
H3

CHART TG. STA2PR SUBROUTINE

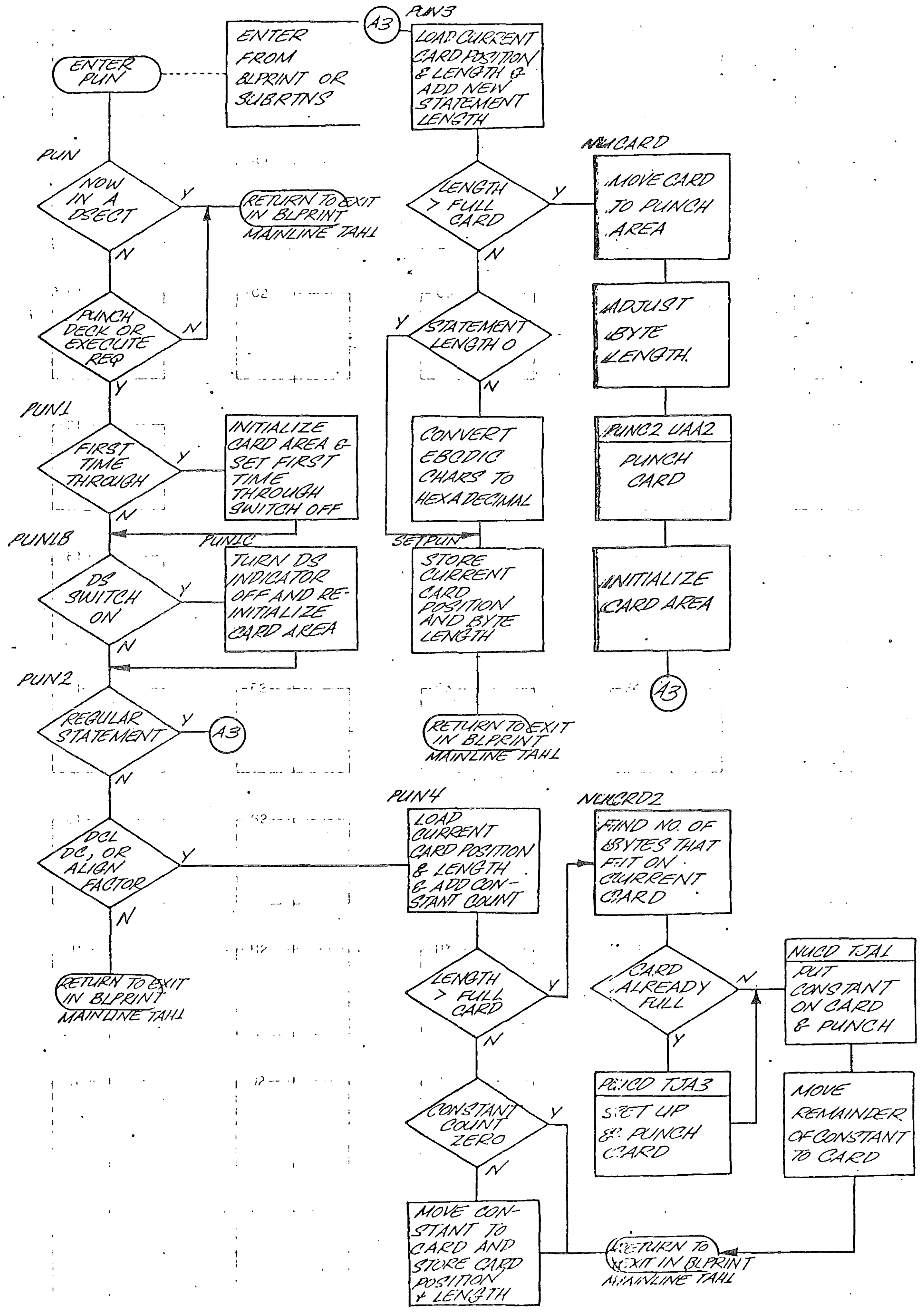

ENTER ASMLOC
NOTE 1
ASMLOC
SET UP TO MOVE AND CONVERT LOCATION COUNTER
ENTER ASMCON
NOTE 2
ASMCON
SET UP TO CONVERT AND MOVE GENERATED CODE
ENTER ROUT
NOTE 3
ROUT
SET UP TO CONVERT AND MOVE MIDDLE OF CONSTANT
ASLP
BREAK HEX BYTE INTO 2 BYTES
FOR HEX DIGITS 1-9, PREFIX WITH HEX F
Z2
FOR HEX DIGITS > 9, SUBTRACT 9 AND PREFIX WITH HEX C
Z2BK
CONVERSION COMPLETE
N
Y
RETURN TO CALLING PT

NOTE 1
ENTRY FROM BLPRINT OR DSECT SUBRTN

NOTE 2
ENTRY FROM ALIG (TBB3)

NOTE 3
ENTRY FROM DC STATEMENT (TCBL, G1, G2)

CHART TH. ASMLOC, ASMCON, ROUT SUBROUTINE



ENTER PUN
ENTER FROM BLPRINT OR SUBRTNS
PUN
NOW IN A DSECT
RETURN TO EXIT IN BLPRINT MAINLINE TAHL
PUNCH DECK OR EXECUTE REQ
PUN1
FIRST TIME THROUGH
INITIALIZE CARD AREA & SET FIRST TIME THROUGH SWITCH OFF
PUN1B
PUN1C
DS SWITCH ON
TURN DS INDICATOR OFF AND RE-INITIALIZE CARD AREA
PUN2
REGULAR STATEMENT
A3
DCL DC, OR ALIGN FACTOR
RETURN TO EXIT IN BLPRINT MAINLINE TAHL
PUN3
LOAD CURRENT CARD POSITION & LENGTH & ADD NEW STATEMENT LENGTH
LENGTH > FULL CARD
STATEMENT LENGTH 0
CONVERT EBCDIC CHARS TO HEXADECIMAL
SETPUN
STORE CURRENT CARD POSITION AND BYTE LENGTH
RETURN TO EXIT IN BLPRINT MAINLINE TAHL
NUCARD
MOVE CARD TO PUNCH AREA
ADJUST BYTE LENGTH
PUNC2 UAA2
PUNCH CARD
INITIALIZE CARD AREA
PUN4
LOAD CURRENT CARD POSITION & LENGTH & ADD CONSTANT COUNT
LENGTH > FULL CARD
CONSTANT COUNT ZERO
MOVE CONSTANT TO CARD AND STORE CARD POSITION & LENGTH
NUCRD2
FIND NO. OF BYTES THAT FIT ON CURRENT CARD
CARD ALREADY FULL
PUCD TJA3
SET UP & PUNCH CARD
NUCD TJA1
PUT CONSTANT ON CARD & PUNCH
MOVE REMAINDER OF CONSTANT TO CARD
RETURN TO EXIT IN BLPRINT MAINLINE TAHL
Y
N

CHART TI. PUN SUBROUTINE

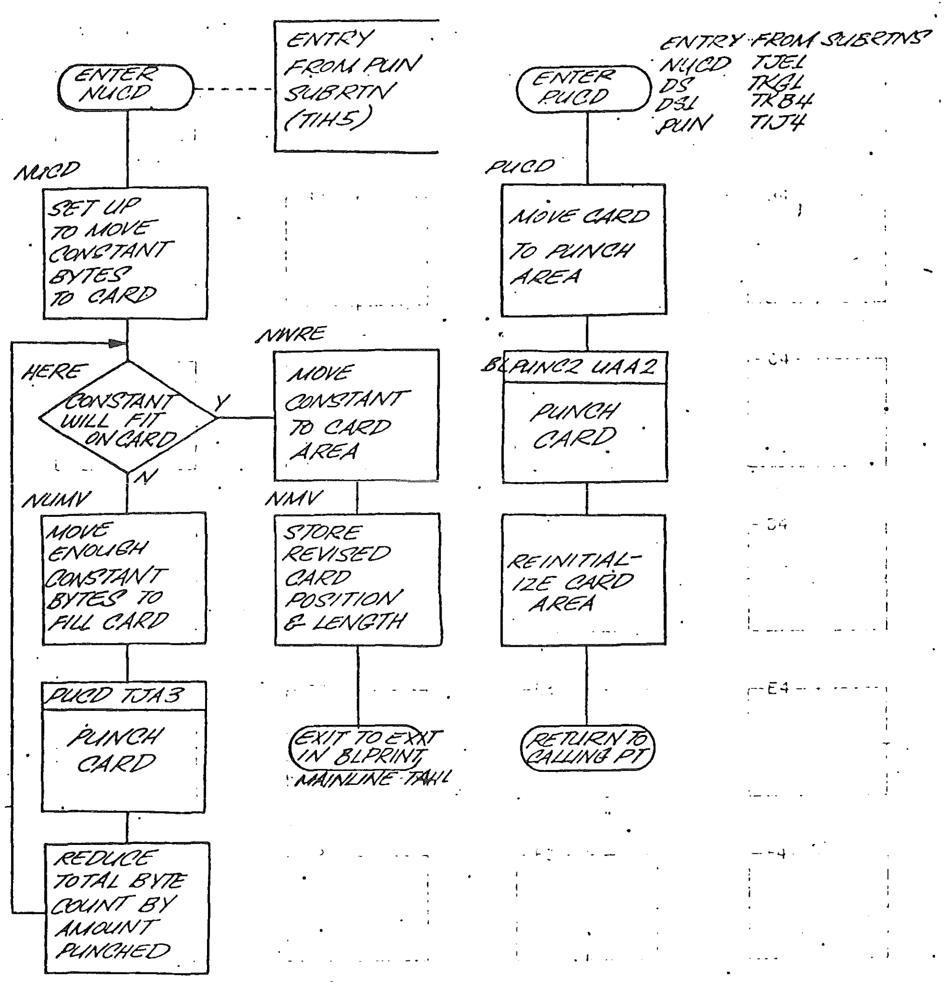

ENTER NUCD
ENTRY FROM PUN SUBRTN (TIH5)
NUCD
SET UP TO MOVE CONSTANT BYTES TO CARD
HERE
CONSTANT WILL FIT ON CARD
Y
N
NWRE
MOVE CONSTANT TO CARD AREA
NUMV
MOVE ENOUGH CONSTANT BYTES TO FILL CARD
NMV
STORE REVISED CARD POSITION & LENGTH
PUCD TJA3
PUNCH CARD
EXIT TO EXXT IN BLPRINT MAINLINE TAHL
REDUCE TOTAL BYTE COUNT BY AMOUNT PUNCHED
ENTER PUCD
ENTRY FROM SUBRTNS
NUCD TJE1
DS TKG1
DS1 TKB4
PUN TIJ4
PUCD
MOVE CARD TO PUNCH AREA
BLPUNC2 UAA2
PUNCH CARD
REINITIALIZE CARD AREA
RETURN TO CALLING PT

CHART TJ. PUCD, NUCD SUBROUTINES

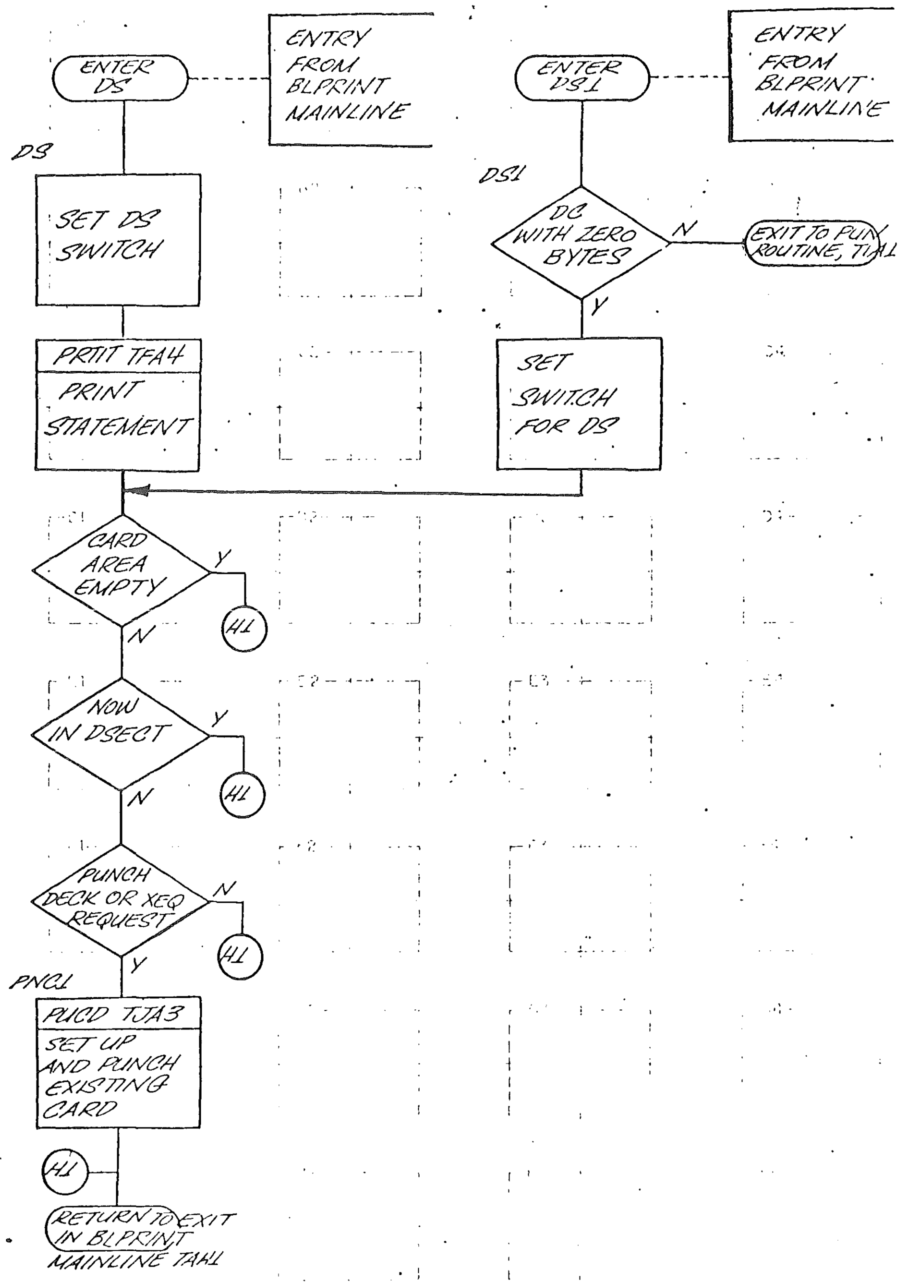


CHART TK. DS, DS1 SUBROUTINES

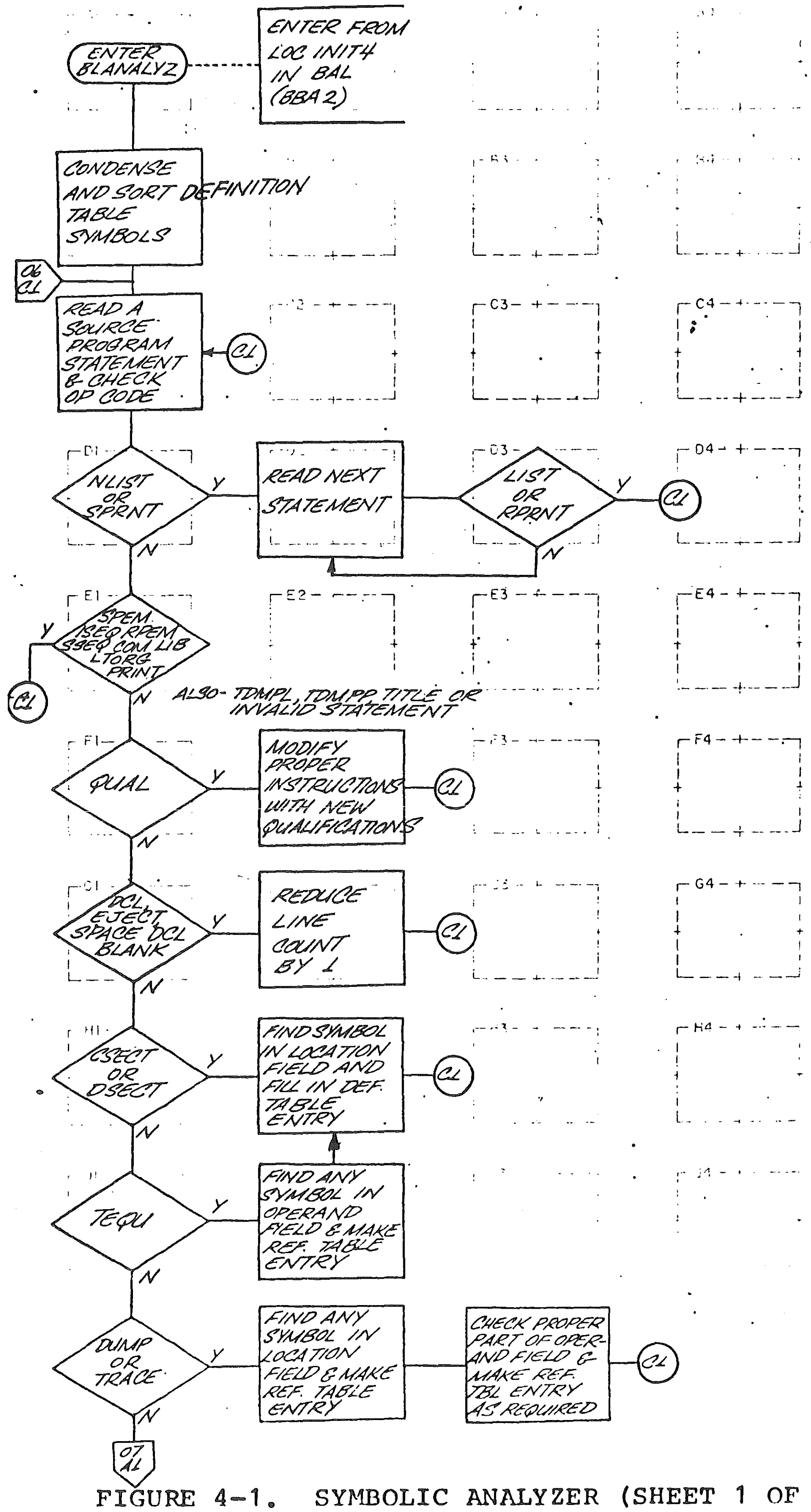
4.0 SYMBOLIC ANALYZER DESCRIPTION

If requested by the programmer, the symbolic analyzer is called after pass 2 to prepare an alphabetic list of all symbols used in the source program. For each symbol, the list gives the number of the line where the symbol was defined as well as the line numbers of all statements referring to that symbol. Multi-defined and systems symbols are flagged. A separate listing of any undefined symbols, including reference line numbers, is printed after the defined symbols list.

The analyzer list results from the merging of two tables: the definitions table and the reference table. The former contains all symbols in the symbol and system symbol tables. During a scan of the source program, line numbers, flags, and a count of number of references are added to the definition table entries. The reference table is built during the same scan, each entry including either the address of a symbol in the definition table and a line number, or an undefined code indicating an entry in the undefined table. A third table, undefined table, contains any undefined symbols that are found.

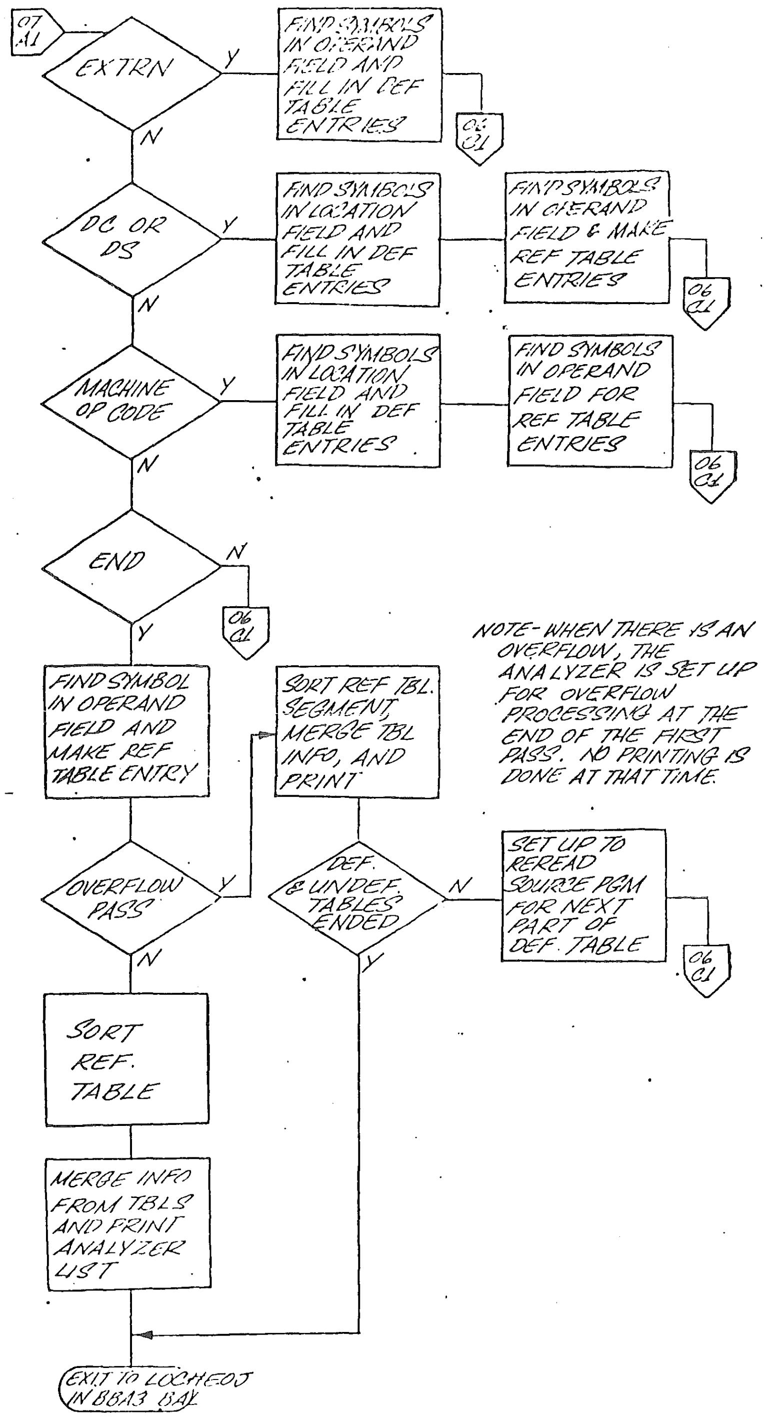
The definition table, followed immediately by the undefined table, starts in lower storage and builds up; the reference table starts in highest storage and builds down. Normally, the analyzer completes processing in one pass. However, if an overflow occurs (i.e., the reference table overlaps the undefined table), the reference table is cancelled but the first pass continues in order to finish the definition and undefined tables. Additional passes are then made to complete the reference table. By adding the number of reference counts given in definition table entries and comparing this to the storage available, the analyzer selects a portion of the definition table and makes another pass over the source program. This time, the reference table is built for the selected portion of the definition table only. After printing, another portion is calculated and another segment of the reference table is generated. This process continues until all symbols and their references have been printed.

Figure 4-1 shows the processing flow of the analyzer.



ENTER BLANALYZ
ENTER FROM LOC INIT4 IN BAL (BBA2)
CONDENSE AND SORT DEFINITION TABLE SYMBOLS
06 C1
READ A SOURCE PROGRAM STATEMENT & CHECK OP CODE
C1
NLIST OR SPRNT
Y
N
READ NEXT STATEMENT
LIST OR RPRNT
Y
N
C1
SPEM ISEQ RPEM SSEQ COM LIB LTORG PRINT
Y
N
C1
ALSO- TDMPL, TDMPP TITLE OR INVALID STATEMENT
QUAL
Y
N
MODIFY PROPER INSTRUCTIONS WITH NEW QUALIFICATIONS
C1
DCL EJECT SPACE DCL BLANK
Y
N
REDUCE LINE COUNT BY 1
C1
CSECT OR DSECT
Y
N
FIND SYMBOL IN LOCATION FIELD AND FILL IN DEF. TABLE ENTRY
C1
TEQU
Y
N
FIND ANY SYMBOL IN OPERAND FIELD & MAKE REF. TABLE ENTRY
DUMP OR TRACE
Y
N
FIND ANY SYMBOL IN LOCATION FIELD & MAKE REF. TABLE ENTRY
CHECK PROPER PART OF OPERAND FIELD & MAKE REF. TBL ENTRY AS REQUIRED
C1
07 A1
C3
C4
D1
D3
D4
E1
E2
E3
E4
F1
F4
G1
G4
H1
H4

FIGURE 4-1. SYMBOLIC ANALYZER (SHEET 1 OF 2)



07 A1
EXTRN
Y
FIND SYMBOLS IN OPERAND FIELD AND FILL IN DEF TABLE ENTRIES
06 C1
N
DC OR DS
Y
FIND SYMBOLS IN LOCATION FIELD AND FILL IN DEF TABLE ENTRIES
FIND SYMBOLS IN OPERAND FIELD & MAKE REF TABLE ENTRIES
06 C1
N
MACHINE OP CODE
Y
FIND SYMBOLS IN LOCATION FIELD AND FILL IN DEF TABLE ENTRIES
FIND SYMBOLS IN OPERAND FIELD FOR REF TABLE ENTRIES
06 C1
N
END
N
06 C1
Y
FIND SYMBOL IN OPERAND FIELD AND MAKE REF TABLE ENTRY
SORT REF TBL. SEGMENT, MERGE TBL INFO, AND PRINT
NOTE- WHEN THERE IS AN OVERFLOW, THE ANALYZER IS SET UP FOR OVERFLOW PROCESSING AT THE END OF THE FIRST PASS. NO PRINTING IS DONE AT THAT TIME.
OVERFLOW PASS
Y
DEF. & UNDEF. TABLES ENDED
N
SET UP TO REREAD SOURCE PGM FOR NEXT PART OF DEF. TABLE
06 C1
Y
N
SORT REF. TABLE
MERGE INFO FROM TBLS AND PRINT ANALYZER LIST
EXIT TO LOCHEOJ IN BBA3 BAX

FIGURE 4-1. SYMBOLIC ANALYZER (SHEET 2 OF 2)

The analyzer consists of 20 logically distinct routines and subroutines which are discussed individually on the following pages. The names and functions of these segments are as follows.

1. BLANALYZ Routine - This routine condenses entries from the symbol and system symbol tables and sorts them in alphabetic order to form the principal part of the definition table.

2. SXXX Routine - This routine reads a source program statement and calls the proper routine for further processing, depending on the operation code.

3. LOC3 Routine - This routine inspects two-character operation code statements and controls the handling of DC and DS statements.

4. LOC1F Routine - This routine decides the handling of three-character operation code statements and oversees END statement processing.

5. LOC2 Routine - This routine decides the handling of five-character operation code statements.

6. LOC1B Routine - This routine processes dump and trace statements.

7. LOC4A Routine - This routine processes CSECT, DSECT, and TEQU statements.

8. DCDS Routine - This routine processes literals and DC and DS statements.

9. EXTE Routine - This routine processes EXTRN statements.

10. LOC10 Routine - This routine identifies symbols in the name and operand fields of all machine operation code statements. It is also used for some pseudo-operation statements.

11. SCH2 Routine - This routine looks up a symbol in the definition (or undefined) table and builds the undefined table.

12. ABKSET Routine - This routine examines an operand field expression, identifies symbols, and makes entries in the reference table.

13. OVERFL Routine - This routine adds line numbers to reference table entries and checks for a reference table overflow.

14. MASTXX Routine - This routine initiates printing, and translates the code used in the undefined table in order to put undefined symbol references in proper order.

15. SRTRT Routine - This routine modifies the sort portion of the ANALY routine for reference table processing.

16. PR Routine - This routine prints the analyzer listing, merging entries from the definition and reference tables.

17. INCSTA Subroutine - called by PR, this subroutine edits an entry line number and converts it to EBCDIC.

18. MAST0 Routine - This routine sets up the analyzer for additional passes when a reference table overflow occurs.

19. MAST3 Routine - This routine resolves any reference table overflow that occurs during an additional pass.

20. XREF Routine - This routine punches XRF cards for input to the XREF subprogram.

4.1 CONDENSE AND SORT ROUTINE (BLANALYZ)

FUNCTION: This routine (Chart SA) begins analyzer processing by building the defintion table with symbols from the system symbol and symbol tables. Definition table entries are filled out (with line numbers, reference counts, and flags) by other analyzer routines. The sort portion of BLANALYZ is also used to place reference table entries in alphabetical order immediately before printing.

ENTRY: This routine has two entry points: BLANALYZ and SRT1. Initial entry is made at BLANALYZ from BAL. SRT1 is entered from the SRTRT subroutine when the reference table is ready for sorting.

OPERATION: If there was symbol table overflow, this routine immediately exits to XEOJBX in the MASTXX routine. The routine checks to see if the XREF option was selected on the $BAL or $JOV card. If so, XRFINT is called. The routine makes a series of loops to condense the symbol and system symbol tables, thus preparing the definition table. The first loop zeros out each symbol itself, and inserts a flag of S. The next loop removes all incomplete or blank entries and all literals from both the system symbol and symbol tables. A third loop zeros out the symbol table entries, except for symbols, and flags then with a blank. At this point, the symbols are ready for sorting.

Symbols are placed in alphabetical order by a merge-replace sort. In this sort, the initial and middle entries in the table are compared and interchanged if they are not in ascending order; then the second entry and the one following the middle one are compared, and so on through the first pass through the table. If the two elements x and y are interchanged, the routine ''backs up'' to compare item y to the items with which x has previously compared, interchanging y with each new item until y has been placed in proper sequence. Another pass over all items is then made with the interval between items compared reduced to one-half of its value on the preceding pass. (This interval is identified as delta on Chart SA.) Succeeding passes through the table are made, in nested loop fashion, each with delta halved from the preceding pass.

The routine uses the ''test value'' to step through the table, comparing items separated by the interval delta. When the end of the table is reached, the test value is incremented and the loop repeated. When the test value exceeds delta, delta is halved and the test value reset for the next pass. When the interval delta is less than the entry length, the sort is complete.

EXIT: The routine can exit to two locations.

1. After building the definition table, the routine exits to the SXXX routine.

2. After completion a reference table sort, the routine transfers control to the analyzer print routine, PR.

COMMENT: This routine is initially set to process 16-byte entries. It is later modified by the SRTRT routine to handle the 8-byte entries of the reference table.

4.2 LOCATE ACCEPTABLE STATEMENTS ROUTINE (SXXX)

FUNCTION: This routine (Chart SB) reads a source program statement, checks the operation code, and determines the action required by the anlayzer.

ENTRY: This routine has three entry points.

1. Location SXXX, from the BLANALYZ routine after the definition table has been set up.

2. Location SE, from the MASTO routine, to reset the start location of the reference table after an overflow has occurred.

3. Location LOC1A, from many sections of the analyzer, in order to fetch the next statement.

OPERATION: When first entered the routine stores the address of the first and last entries in the definition table. The latter address is saved in two locations, since the contents of one will be altered if additional analyzer passes are needed. The routine also sets the address of the first and last entries in the undefined table, located 16 bytes after the last entry in the definition table. The address of the first entry in the reference table, which starts at the highest storage location available, is developed and stored along with addresses for the definition table.

The routine then calls the BLIOGET2 routine to read a source program statement in canonical form. The first byte of the statement is checked to decide whether a comment, ignore, or literal reference record is present. Ignore, literal reference, JOVIAL, and comment statements are disregarded, although the line counter is incremented for a comment. Certain pseudo-operation statements that require no further handling are treated the same as comments. A QUAL statement, if valid, results in storing the new symbol qualification in the LOC4A, LOC10, LOC1B and ABKSET routines, so that the qualification character may be moved with the rest of the symbol to which it applies.

All other types of statements cause a branch to the proper routine, as long as the NLIST switch remains off. This switch is set when an NLIST statement appears; all statements are then ignored until a LIST is found.

EXIT: This routine may exit to one of six locations, depending on the operation code of the statement read.

1. To the LOC3 routine, used for all two-character operation code statements.

2. To the LOC1F routine, taken for all three-character operation code statements.

3. To the LOC2 routine, taken for all statements that have operation codes more than four characters long.

4. To location LOC1A1, in the LOC10 routine, used to process the operand field of a TEQU statement.

5. To the LOC1B routine, taken for a dump or trace statement.

6. To the LOC10 routine, used for all four-character machine operation code statements.

4.3 TWO-CHARACTER OPERATION CODE ROUTINE (LOC3)

FUNCTION: This routine (Chart SC) controls the processing of statements with two-character operation codes.

ENTRY: Entry is at location LOC3 from the SXXX routine, made when a two-character operation code is recognized.

OPERATION: Any two-character operation code other than a DC or DS results in an immediate exit to the LOC10 routine.

For a DC or DS, the routine first modifies the LOC10 routine so that it checks only the name field of a statement, and then transfers to the LOC10 routine. Upon return, the DCDS routine is set up for DC and DS processing and called to check the statement operand field.

EXIT: Exit is to location LOC1A in the SXXX routine after DC or DS processing. Otherwise, control is transferred to LOC10.

4.4 THREE-CHARACTER OPERATION CODE ROUTINE (LOC1F)

FUNCTION: This routine (Chart SD) completes the processing of DCL, DC blank, COM, and LIB statements, and sets up for END statement handling. Routine LOC1F passes all other three-character operation code statements to the LOC10 routine.

ENTRY: This routine is entered at location LOC1F from the SXXX routine whenever a three-character operation code is read.

OPERATION: A DCL or DC blank operation code causes the routine to decrement the line counter and exit to the SXXX routine for the next statement. In effect, these two types are ignored. If a COM or LIB is found, exit is also to SXXX, but without decrementing the line counter. All other three-character operation code statements, except END, result in an exit to the LOC10 routine.

When an END statement is read by LOC1F, a switch is set in LOC10 so that return is made to LOC1F2 after any symbol in the statement has been processed. Two tests are then made to determine the next action. If a reference table overflow occurred during the first pass, a branch is taken to set up for additional analyzer passes; if the analyzer is now making an additional pass, a branch is made to initiate printing of the current segment of the reference table. If both tests are negative (the usual condition), printing of the analyzer listing is started.

EXIT: This routine may exit to any of five locations.

1. For DC blank, DCL, COM, and LIB statement, exit is to LOC1A in the SXXX routine to fetch the next statement.

2. For all other three-character operation code statements, exit is to the LOC10 routine.

3. If no reference table overflow has occurred, the return from END statement processing results in an exit to the MASTXX routine to initiate printing of the analyzer listing.

4. If a reference table overflow has occurred during this pass, return from END statement processing results in an exit to the MAST0 routine, which sets up for additional passes.

5. During additional passes, return from END statement processing causes an exit to location MAST1 in the MAST0 routine to print the current segment of the reference table.

4.5 FIVE-CHARACTER OPERATION CODE ROUTINE (LOC2)

FUNCTION: This routine (Chart SE) controls the action taken for statements with operation codes of five or more characters.

ENTRY: Entry to this routine is at location LOC2. It is entered from the SXXX routine when an operation code longer than four characters is read.

OPERATION: The routine uses a series of comparisons to determine the proper exit for each statement. If an NLIST is found, LOC2 sets an indicator in the SXXX routine so that subsequent source program statements are ignored until a LIST occurs. If a START is found, a switch is set to ignore all operands.

EXIT: There are six possible exits from this routine.

1. To the LOC10 routine, taken for an unidentified operation code or an operation code with more than five characters.

2. To the LOC4A routine, taken for a CSECT or DSECT statement.

3. To the EXTE routine, taken for an EXTRN.

4. To location LOC1A, in the SXXX routine, taken for an NLIST, SPACE, EJECT, LTORG, TDMPL, TDMPP, TITLE, or PRINT.

5. To location LOC1BT, in the LOC1B routine, taken for a TRAC(x).

6. To the LOC1B routine, taken for a DUMP(x).

4.6 DUMP-TRACE ROUTINE (LOC1B)

FUNCTION: This routine (Chart SF) isolates symbols in the name and operand fields of dump and trace statements.

ENTRY: According to the specific operation code being processed, either of two entry points may be used.

1. LOC1BT, used by the LOC2 routine when a trace statement is found.

2. LOC1B, used by the LOC2 routine for DUMP(x) statements and by the SXXX routine for DUMP statements.

OPERATION: A symbol in a trace statement should appear after the first comma in the operand field. Thus, the trace entry point (LOC1BT) sets the comma tally to one. For all dump statements, symbols are assumed to be after the second comma, and a two is loaded in the comma tally. Since a DUMPR statement requires special handling (after the fourth comma, two fields are skipped and the symbol following must be processed), the routine will take the branch to location LOC33.

A symbol found in the name field of a dump or trace statement is considered to be a reference only. In this case, the symbol, its qualification, and the current line number are readied for the reference table. Before calling the SCH2 routine to check that the symbol was defined, LOC1B sets up SCH2 to return through the ABKSET routine. There the reference table entry will be completed before LOC1B begins its examination of the statement operand field.

After return from ABKSET, or if there is no symbol in the name field, LOC1B scans the operand, reducing the comma tally for each comma found. As soon as the tally becomes zero, a pointer is set at the next byte and the routine branches to the LOC10 routine to process the remainder of the statement.

For a DUMPR statement, a switch is turned on in the LOC10 routine so that a return is made to LOC1B (at location LOCRC) after a single symbol is processed. By means of a decremented tally, LOC1B makes two calls to LOC10. Then it bypasses two fields and branches (with the return switch off) to LOC10 to process the remainder of the statement.

EXIT: This routine can exit to either of two locations.

1. Location EX2, in the LOC10 routine, used when LOC1B has completed examination of a statement.

2. Location LOC1A, in the SXXX routine, taken if a terminating blank is embedded in the operand field.

4.7 CSECT-DSECT-TEQU ROUTINE (LOC4A)

FUNCTION: This routine (Chart SG) examines CSECT, DSECT, and TEQU statements and, as appropriate, moves information to a definition table entry or sets up an entry for the reference table.

ENTRY: The routine has three entry points.

1. Location LOC4A, from the LOC2 routine when a CSECT is detected.

2. Location LOC4, from the LOC10 routine for a TEQU statement.

3. Location LOC4A0, from the LOC2 routine when a DSECT is detected.

OPERATION: The routine checks only the name field of the input statement, expecting a valid symbol definition. If this proves true after a call to SCH2, LOC4A moves the current line number to the definition table entry for the symbol and exits.

If the symbol was previously defined (i.e., the table entry already has a line number), operation depends on whether a reference table overflow has occurred.

If there was an overflow, the symbol is treated as a reference as long as it does not have the same line number as the previous definition. When there is no overflow, redefinitions in a TEQU statement are handled as references; they are not an error. Previous definition of a CSECT or DSECT symbol is also permissible, provided the first definition was in a CSECT or DSECT statement. If this is the case, the new definition is treated as a reference. But if the earlier definition was in any other type of statement, a multi-defined or multi-redefined flag is set before preparing a reference entry. For a DSECT, the QUAL character is ignored to provide proper handling of EXTRNed DSECTs in qualified code.

EXIT: This routine has two exits.

1. Location LOC1A, in the SXXX routine, taken to get the next statement.

2. Location ABK23, in the ABKSET routine, taken to set up an entry for the reference table. Routine LOC4A supplies location LOC1A as a return address for ABKSET.

ERRORS: The routine sets an error flag (M or N) if, during a nonoverflow pass, CSECT or DSECT statement has a symbol that was defined before in another type of statement. The legend printed for these flags appears in the PR routine description.

4.8 DCDS Routine

FUNCTION: This routine (Chart SH) isolates an expression in a literal or in the operand field of a DC or DS statement for the ABKSET routine.

ENTRY: The routine has one entry point, location DCDS. It is entered from the LOC3 routine for a DC or DS and from the LOC10 routine for a literal.

OPERATION: The routine begins by searching for the start of an expression, indicated by a left parenthesis, exiting if it encounters a blank or comma. When a left parenthesis is found, ABKSET is called to process any symbols in the expression and to make the necessary entries in the reference table. Upon return, the pointer will be at the expression terminator. For a literal, signaled by a value that LOC10 put in a general register, a comma results in a return to the calling routine. For a DC or DS, the loop through ABKSET continues until an expression terminator other than a comma is found.

EXIT: Exit is to the calling routine.

4.9 EXTRN ROUTINE (EXTE)

FUNCTION: This routine (Chart SI) controls the processing of symbols in the operand field of EXTRN statements. Each symbol is identified separately for the LOC10 routine by setting a pointer to the first byte of the EXTRN statements. Each symbol is identified. Symbols are definitions rather than references.

ENTRY: The only entry point is location EXTE, entered from the LOC2 routine when an EXTRN is found. The calling routine supplies the location of the first byte in the operand field.

OPERATION: The routine searches the operand field for symbols, using an 8-byte maximum length as the criterion for a valid symbol. The routine immediately exits if the first byte of the operand is blank. Otherwise, it forms a loop to find a terminator: a blank, comma, or period (indicating qualification). Since processing of a symbol is identical with the LOC10 routine handling of a name field, a blank or comma results in a call to LOC10, which is set to return after name field processing. Since a blank also marks the end of the operand, the EXTRN exits upon return. A period causes the routine to strip the qualification before resuming standard processing.

When more than eight bytes are scanned before encountering a terminator, the symbol is assumed invalid. If a comma appears after the invalid symbol, the routine loops back to start the next scan, thus ignoring the invalid symbol.

EXIT: This routine exits to location LOC1A in the SXXX routine.

4.10 LOCATE SYMBOLS ROUTINE (LOC10)

FUNCTION: This routine (Charts SJ, SK) isolates symbols in the name and operand fields of statements with machine operation codes. It also helps process dump, trace, EXTRN, END, TEQU, DC, and DS statements.

ENTRY: The routine has four entry points.

1. LOC10 from the SXXX, LOC1, LOC1F, and LOC2 routines for a machine operation code or for a DC or DS statement.

2. LOC1A1, from the LOC1F routine for an END statement, and from the SXXX routine for a TEQU.

3. EX2, from the LOC1B routine for a dump or trace.

4. LOC10A, from the EXTE routine, which sets up one symbol to be processed as if it were in the name field of a statement.

OPERATION: Name field processing is bypassed if the field has no symbol or if the analyzer is in an additional pass. (All definition entries are completed in the first pass, regardless of reference table overflow. In additional passes, to save time, definition activity is inhibited.)

A symbol discovered in the name field is set up for the SCH2 routine. Upon return, if SCH2 did not find the symbol in the definition table, LOC10 increments the reference count of the entry in the undefined table, moves the definition information to the reference table, and calls the OVERFL routine to complete the reference table entry.

If the symbol was found, a test is made to decide which flag should be put in the definition table entry. A multi-defined symbol is flagged with an R if a system symbol or with an M if a source program symbol; a properly defined symbol is flagged with either a T (system symbol) or D. These flags serve as signals for other analyzer routines.

The segment of LOC10 beginning at location LOC1AB examines an operand field for symbols, and thus looks for possible entries for the reference table. There are two special cases, however: a DC or DS statement is bypassed because of a switch set by the LOC3 routine; a DUMPR statement results in a return to the LOC1B routine after a single operand symbol is handled.

For other statements, LOC10 begins a series of comparisons to isolate expressions, continuing until the first blank signals the end of the operand field. Each time an expression is found, the routine branches to ABKSET so that possible symbols may be processed for the reference table. An equal sign, indicating a literal, results in a branch to the DCDS routine, which is first set up for literal processing.

EXIT: This routine may exit to one of six locations.

1. If an EXTRN is being processed, the routine exits to the EXTE routine after processing one symbol.

2. To the LOC3 routine, for a DC or DS after name field processing.

3. To LOCRC in the LOC1B routine, taken if a DUMPR statement is being processed.

4. If an END statement is in process, exit is to location LOC1F2 in the LOC1F routine.

5. If a machine operation code statement was processed, the routine exits to LOC1A in the SXXX routine.

6. To location LOC4, in the LOC4A routine, for a TEQU statement.

ERRORS: When the routine detects a multi-defined symbol, it flags the definitions table entry with an M or an R. These flags will appear in the analyzer listing.

4.11 SEARCH ROUTINE (SCH2)

FUNCTION: This routine (Charts SL, SM) finds the address of a symbol in the definition or undefined table. It also builds the undefined table to include any symbol not in the definition table.

ENTRY: The routine has one entry point, location SCH2, which may be entered from the LOC1B, LOC4A, LOC10, and ABKSET routines. The calling routine will indicate whether SCH2 is to place the address of the matching definition table entry in the next available reference table location or in a specific storage area.

OPERATION: In the first analyzer pass, this routine searches the defintion table and, if required, the undefined table. If a match for a symbol is not found, a new entry is made in the undefined table. This table, like the definition table, is always completed during the first pass.

In additional passes, operation is limited to a search of the definition table. The undefined table, if present, is then treated just like the definition table after all definition table entries and references are printed.

The routine begins its search at the mid-point of the definition table, comparing the entry with the symbol sought to decide which half of the table to scan.

The routine then sets starting and ending locations for that portion of the table and continues this process until it has found a match or exhausted the table. As soon as a match is found, the routine stores the entry address as directed by the calling routine and exits.

If the limit is reached in the selected portion of the table and there is no match, the symbol is undefined, and the first entry is made in the undefined table. This entry is given a code number of 1; a code number is used rather than an address, since the address may change as new entries are added. (Each new undefined symbol is assigned the next higher code number until the table limit of 255 is reached.) After generating the entry, the routine places the undefined code in the next available reference table location (if no overflow has occurred) and exits.

For subsequent undefined symbols, the routine enters the same loop used in definition table search, but with the pointer now set to the undefined table. When a match is found, the undefined code is moved to the reference table; if there is still no match, the new undefined symbol is entered in its proper alphabetic position in the undefined table.

If a reference table overflow occurs, all subsequent reference table entires for undefined symbols will carry an undefined table address rather than an undefined code. Since the undefined table is complete at the end of the first pass and its entries are thus in proper alphabetic order, normal table addresses can be used.

EXIT: Exit is to the calling point.

4.12 ABKSET ROUTINE

FUNCTION: This routine (Chart SN) develops the reference table and increments the reference count in defintion and undefined table entries.

ENTRY: The routine has three entry points.

1. Location ABKSET, from the LOC10 and DCDS routines.
2. Location ABK23, from the LOC1B and LOC4A routines.
3. Location ABK01, from the MAST3 routine.

The calling routine supplies the address of the expression to be processed.

OPERATION: The routine calls BLBRKUP to separate the input expression into elements and operators, to find the number of characters in each element, and to locate the expression terminator.

Routine ABKSET processing starts with a test of the diagnostic code returned by BLBRKUP. If the expression proves so long or complex that the entire statement is invalid, the routine exits at once to get the next statement; immediate return is made to the address provided by the calling routine when any other kind of error is indicated. Otherwise, the first character in an element, translated by an internal table, determines the operations to be performed. The path taken for various types of first characters are as follows.

First Character	Path
Number	Branch to ABK1 and then to ABK00. In effect, this element is ignored.
Alphabetic (except X, C, S, T, L) or $	Branch to ABK21, where the symbol is set up for the SCH2 routine.

First Character	Path
Blank or Quote	Branch to ABK3, where a scan is made to the next quote. The characters read are ignored, and the routine loops to ABK00 to process the next element.
X or C	Branch to ABK4, where a check is made for quotes in the second byte. If found, processing is the same as for an initial quote. If quotes are not detected, the path for alphabetic processing is taken.
S, T, or L	Processing continues with a test for quotes in the second byte. If present, the routine skips to the next character, assumed to be the start of a symbol, before continuing with alphabetic processing.
Any other character.	Treated the same as a number.

Before calling the SCH2 routine, ABKSET sets up the symbol it has isolated and checks whether special qualification (signaled by a period immediately after the symbol) is involved. If so, the qualification is placed after the symbol, overlaying any current qualification, before transfer of control. Routine SCH2 then scans the definitions table and, if necessary, the undefined table to find the table entry matching the symbol and moves that entry's address or undefined code to the next available location in the reference table.

Upon return, ABKSET makes a series of tests before incrementing the reference count for the current symbol (in the definition or undefined table) and before calling the OVERFL routine to make the reference table entry final. Essentially, these tests prevent any additions to the definition and undefined tables after the first analyzer pass. If an overflow occurs in the first pass, the tests forbid further entries in the reference table but allow the definition and undefined tables to be completed. The reference table will then, of course, be built in additional passes.

EXIT: The routine normally exits to the address supplied by the calling routine. At exit, the address of the expression terminator is loaded into a general register for the use of the calling routine. If the return from BLBRKUP shows that the entire input statement is invalid, the routine exits to LOC1A in the SXXX routine to read the next statement.

4.13 OVERFL ROUTINE

FUNCTION: This routine (Chart SO) places a line number in the current reference table entry, develops the location for the next entry, and checks for a reference table overflow. When an overflow occurs, the routine sets switches to prevent further building of the reference table and sets up for a transfer to the MAST0 routine when the END statement is read.

ENTRY: There is one entry point, location OVERFL, which is entered from the ABKSET routine whenever a reference table entry is ready. It is also entered from the LOC10 routine when a reference to an undefined symbol is being processed.

EXIT: This routine has two exits.

1. The usual exit to the calling point.

2. If an overflow occurs during an additional pass, the routine transfers control to the MAST3 (double overflow) routine.

COMMENT: No reference table entry is final until this routine has supplied the next reference entry location. If no new location is given, the next entry moved to the table will wipe out the previous one.

4.14 REFERENCE-TABLE-CODE TRANSLATE ROUTINE (MASTXX)

FUNCTION: This routine (Chart SP) initiates printing of the analyzer listing (i.e., the reference table has not overflowed). If necessary, it also translates undefined symbol codes, used instead of addresses in the undefined table, so that sorting will place references to undefined symbols in proper order. The analyzer end-of-job sequence is also included in this routine.

ENTRY: This routine has two entry points.

1. Location MASTXX, from the LOC1F routine after END statement handling.

2. Location XEOJB, from the PR or MAST0 routine after printing. Entry is from MAST0 only if the reference table overflowed during analyzer processing.

OPERATION: If there are no undefined symbols, the routine immediately calls SRTRT to initiate printing. Otherwise, it builds an internal table to equate each undefined code with the alphabetical position of its symbol in the undefined table. (Undefined symbols are in alphabetic order, but codes are assigned in the order the symbols were encountered.) Once this translation table is complete, the reference table is read and each undefined code is replaced with its corresponding sequence number. Control then passes to the SRTRT routine. Before the final EXIT to MASTER the routine checks to see if the XREF option was selected. If so, the XRFTRL subroutine is called.

EXIT: This routine has two exits.

1. To the SRTRT routine to begin printing.

2. To MASTER, indicating the end of analyzer operations.

ERRORS: The following message is printed if more than 255 undefined symbols appeared in the source program:

```
THE UNDEFINED SYMBOLS LIMIT HAS BEEN EXCEEDED.
THE FIRST 255 ARE LISTED.
```

COMMENT: Translation of undefined codes is unnecessary in additional passes. Since the undefined table is always completed during the first pass, actual table addresses can be used to identify references to undefined symbols.

4.15 PREPARE FOR SORT ROUTINE (SRTRT)

FUNCTION: This routine (Chart SQ) sets up the sort portion of the ANALY routine for reference table processing.

ENTRY: Location SRTRT is the only entry point. Usually, it is entered from the MASTXX routine when the analyzer tables are complete. In case of overflow, entry is from the MAST0 routine to print a segment of the analyzer listing. In the unlikely event of a second overflow, entry is made from the MAST3 routine to resolve the overflow.

OPERATION: The routine modifies BLANALYZ to sort reference table entries into proper sequence. After saving registers for PR, SRTRT alters the length values stored in BLANALYZ to 8 bytes, the length of reference table entries; and sets switches to permit duplicate entries, (valid in the reference table) and to cause BLANALYZ to exit directly to PR. Reference table starting and ending locations and register increment values are also established.

Because of a first-time through indicator, SRTRT bypasses its switch-setting and length-modifying operations in subsequent calls.

EXIT: This routine exits to location SRT1 in the ANALY routine.

4.16 PRINT ROUTINE (PR)

FUNCTION: This routine (Charts SR, SS, ST) places the analyzer listing on system output. Routine PR develops the listing by merging and editing entries from the definition, reference, and undefined tables.

ENTRY: The routine has one standard entry point and three special entry points that are used during double-overflow processing. Location PR is the standard entry from the ANALY routine after reference table sorting. The special entries are as follows.

1. Location PRSW1, from the MAST3 routine, when a double-overflow is discovered.

2. Location PR3, from the MAST3 routine, when the preceding attempt to resolve the double-overflow has failed and a partial line is stored in the print area.

3. Location PR21, from the MAST3 routine, when the preceding attempt to resolve the double-overflow has failed and the print area is ready for a new line.

OPERATION: This routine may operate in three situations.

1. In standard processing, the MASTXX routine makes one call to PR (through the SRTRT and BLANALYZ routines). Printing in this case is the last analyzer operation; the entire listing is printed, and exit is to end-of-job (XEOJB in MASTXX).

2. When a reference table overflow has occurred, the MAST0 routine calls PR (through SRTRT and BLANALYZ) at the end of each additional pass to print a segment of the analyzer listing.

3. When a double-overflow appears, the MAST3 routine immediately calls PR (through SRTRT and BLANALYZ), and the two routines interact until the special overflow condition is resolved.

STANDARD PRINTING: The routine begins by saving the starting and ending locations of the definition table. After finding the first valid entry (i.e., one that contains a definition line number or a nonzero reference count), it takes a first-time branch to set up column headings for defined symbols and ejects the listing to a new page.

The routine next examines the flag field of the entry. A flag of D marks a properly defined symbol, and no flag is moved to the print line. If the flag is greater, a diagnostic switch is set in MASTER and the entry flag is moved to the print area. Usually, this is either an R (multi-redefined system symbol) or M (multi-defined symbol). If the flag is T, it is converted to an S (for redefined system symbol), which will be accepted by MASTER as a proper entry.

The INCSTA subroutine (Chart SU) is now called to edit the entry line number and convert it to EBCDIC, after which it is moved to the print area, followed by the symbol and its qualification, (if any). If a test of the entry reference count shows there are no references for the symbol, the contents of the print area are placed on system output, and the routine loops back to fetch the next definition table entry. When references are indicated, the line numbers of the corresponding reference table entries are edited, converted to EBCDIC, and moved to the print area. Up to nine reference line numbers can be printed on the same line with the symbol. Any additional references are then printed on subsequent lines without repeating the flag, definition line number, symbol, and qualification information. Before a line is printed, the routine checks to see if the XREF option is active. If so, Subroutine XRFBEG is called to set up the symbolic card.

When entries in the definition table are exhausted, the routine checks for undefined table entries, exiting to end-of-job if there are none. Otherwise, it sets up for the undefined table, prints page and column headings for the undefined symbol listing, then loops back to begin printing undefined symbols and their references. The process is the same as for defined entries, except that the flag and the definition line numbers are not printed.

PRINTING BY SEGMENTS: If the reference table overflowed, nothing is printed at the end of the first analyzer pass, and only a segment of the definition table (or undefined table), along with its corresponding references, is printed at the end of each subsequent pass. In this situation, the starting and ending locations of the definition table are set by the MASTO routine. PR operation for a segment is the same as in the standard printing, but when the segment is finished, exit is to the MAST0 routine. Routine MAST0 will determine when the analyzer listing is complete.

PRINTING IN A DOUBLE-OVERFLOW SITUATION: The operation of PR in a double-overflow situation is controlled by the MAST3 routine. (See the description of that routine for the procedure used.)

EXIT: Exit is to the address supplied by the calling routine. In standard processing, return is to the end-of-job portion of the MASTXX routine. In segment processing, return is to the calling point in the MAST0 routine. All exits in double-overflow handling are to the MAST3 routine.

ERRORS: When an M, T (printed out as an S), or R flag is detected in a definition table entry, the flag is moved to the listing and a diagnostic switch is set in MASTER. Before the assembler returns control to the utility monitor, it will print a flag legend.

M	ANALYZER HAS FOUND A SYMBOL TO BE MULTI-DEFINED.
S	ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH WAS REDEFINED IN THE PROGRAM. (This flag is for information; it is not an error.)
R	ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH IS MULTI-REDEFINED.

4.17 CONVERT TO EBCDIC SUBROUTINE (INCSTA)

FUNCTION: This subroutine (Chart SU) converts a binary number to deciaml (EBCDIC representation) for printing.

ENTRY: The one entry point, location INCSTA, is entered from various segments of the PR routine.

EXIT: Exit is to the calling point in the PR routine.

4.18 DOUBLE-OVERFLOW ROUTINE (MAST3)

FUNCTION: This routine (Chart SW) manipulates the PR routine to resolve a reference table overflow during an additional pass. This double-overflow can happen only when the references for a single symbol exceed the area available for the entire reference table segment.

ENTRY: The routine has three entry points, which are used in sequence.

1. Location MAST3, from the OVERFL routine when the double-overflow is first detected. (This occurs as the ABKSET routine tries to add and entry to a full reference table.)

2. Location MAST31, from the PR routine just after it sets up for printing.

3. Location MAST34, entered from PR after it has performed the operation designated by MAST3.

OPERATION: When first called, the routine saves registers, sets two double-overflow switches in the PR routine, and then calls SRTRT to initiate printing. The saved registers preserve the exact conditions under which ABKSET began processing the current reference; later, they will be restored to restart ABKSET.

Once PR is ready for printing, it encounters the first double-overflow switch and exits to MAST31. In turn, MAST3 passes control to location PRSW1 in PR to print all references now in the reference table. As soon as this is done, PR returns to MAST34, where MAST3 resets the start of the reference table, restores registers, and exits to ABKSET to resume processing the reference that caused the overflow. (At this point, the print area may contain a partial line, but this will be handled within the PR routine by the setting of the second double-overflow switch when normal additional-pass printing starts.)

If another overflow appears, the initial call and the first entry from the PR are the same as before. But MAST3 now branches to location MAST36, where the contents of the print area decide the next action. If there is information in the print area, the routine exits to location PR3 to fill out and print the line. If the print area is clear, the routine exits to location PR21 to set up a new line. In either case, printing continues until the current reference table entries are exhausted. Upon final return from PR, MAST3 exits as described below.

EXIT: Final exit is always to location ABK01 in the ABKSET routine. Other exits that may be used are as follows.

a. To the SRTRT routine to initiate printing.

b. To location PRSW1 in the PR routine to set up and print the initial line(s) for the entry.

c. To location PR3 in PR to complete and print a partial line stored in the print area.

d. To location PR21 in PR to set up a new line when the print area is clear.

4.19 REFERENCE TABLE OVERFLOW ROUTINE (MASTO)

FUNCTION: This routine (Chart SY) is called when the source program END statement is read and an overflow has occurred. On the first call, the routine sets up the analyzer for additional passes so that it can build the reference table in segments. At the end of each additional pass, the routine initiates printing of the completed portions of the definition and reference tables and sets up for the next pass.

ENTRY: Entry to this routine is at location MASTO from the LOC1F routine when the END statement is in process and an overflow is indicated. When the END statement is encountered at the end of each additional pass, entry is at location MAST1.

OPERATION: As soon as overflow is detected, the reference table becomes void and its area is surrendered to the undefined table. The filling of entries in the definition and undefined tables continues so that when the END statement appears at the end of the first analyzer pass, these two tables are complete and their length is fixed. Therefore, MASTO can calculate the exact storage, and from it the maximum number of entries available for the reference table. Then, starting at the first entry in the definition table, the routine adds up reference counts until they exceed the number of possible entries, and sets a new definition table ''end'' address at the entry just before the one causing the excess. The reference table will now be built for the selected portion of the definition table only. When this segment of the analyzer listing is printed, the next portion is established, and the process continues until all symbols and references are printed.

To prepare for additional passes, MASTO sets seven switches: two in SCH2 to prohibit further entries in the undefined table and to permit the reference table to be built; two in ABKSET, one to inhibit building of the definition table, the other to allow reference table entries; one in the LOC10 routine to inhibit definition table building; one in OVERFL so that a branch is taken to the MAST3 routine if another overflow occurs; and one in LOC1F, causing a return to MASTO (at location MAST1) the next time an END statement is read.

EXIT: This routine may take either of two exits.

1. Location SE, in the SXXX routine, taken to start each additional pass.

2. Location XEOJB, in the MASTXX routine, taken when printing has been completed. This exit is also used if there is no storage available for the reference table.

4.20 PUNCH XRF CARDS ROUTINE (XREF)

This routine punches XRF reference cards when requested by the XREF option on the $BAL Control Card. The routine consists of four subroutines.

1. XRFINIT Subroutine - This subroutine initializes for the XREF function.

2. XRFBEG Subroutine - This subroutine processes each symbol reference and builds the XRF symbol card image.

3. XRFPCH Subroutine - This subroutine punches cards for the XREF function.

4. XRFTRL Subroutine - This subroutine provides termination of the XREF function.

4.20.1 XRFINIT Subroutine

FUNCTION: This subroutine (Chart SZ) performs necessary initialization for the XREF function.

ENTRY: The subroutine has only one entry point at XRFINIT. The entry is made from BLANALYZ if the XREF option was requested.

OPERATION: This subroutine first checks to see the type of assembly being made. If the assembly is for a Compool Segment, no switches are set and return is made to the BLANALYZ routine. If serious errors were encountered during the Assembly, no switches are set and return is made to BLANALYZ. Otherwise a switch is set to indicate that XRF cards should be punched. The XRF header card is then prepared and subroutine XRFPCH is called to punch the header card. When the header card has been punched, return is made to the calling point.

EXIT: Exit from this routine is made to the calling point in the BLANALYZ routine.

4.20.2 XRFBEG Subroutine

FUNCTION: The purpose of this subroutine (Chart SZ) is to build XRF symbol cards.

ENTRY: Entry is made at XRFBEG from the PR routine whenever the Analyzer is ready to set up a line of print with a referenced symbol.

OPERATION: The subroutine first checks to see that the referenced symbol is a library routine name or a Compool data name. If not return is made to the PR routine. The symbol is put in the XRF card image and, if the card is full, Subroutine XRFPCH is called to punch the card.

EXIT: Exit is made to the calling point in the PR routine.

4.20.3 XRFPCH Subroutine

FUNCTION: This subroutine (Chart SZ) punches XRF cards.

ENTRY: The routine may be entered at XRFPCH from the XRFINIT subroutine, the XRFBEG subroutine, or the XRFTRL subroutine.

OPERATION: This routine places a sequence number on the card to be punched and then punches the card.

EXIT: Exit is made to the calling point.

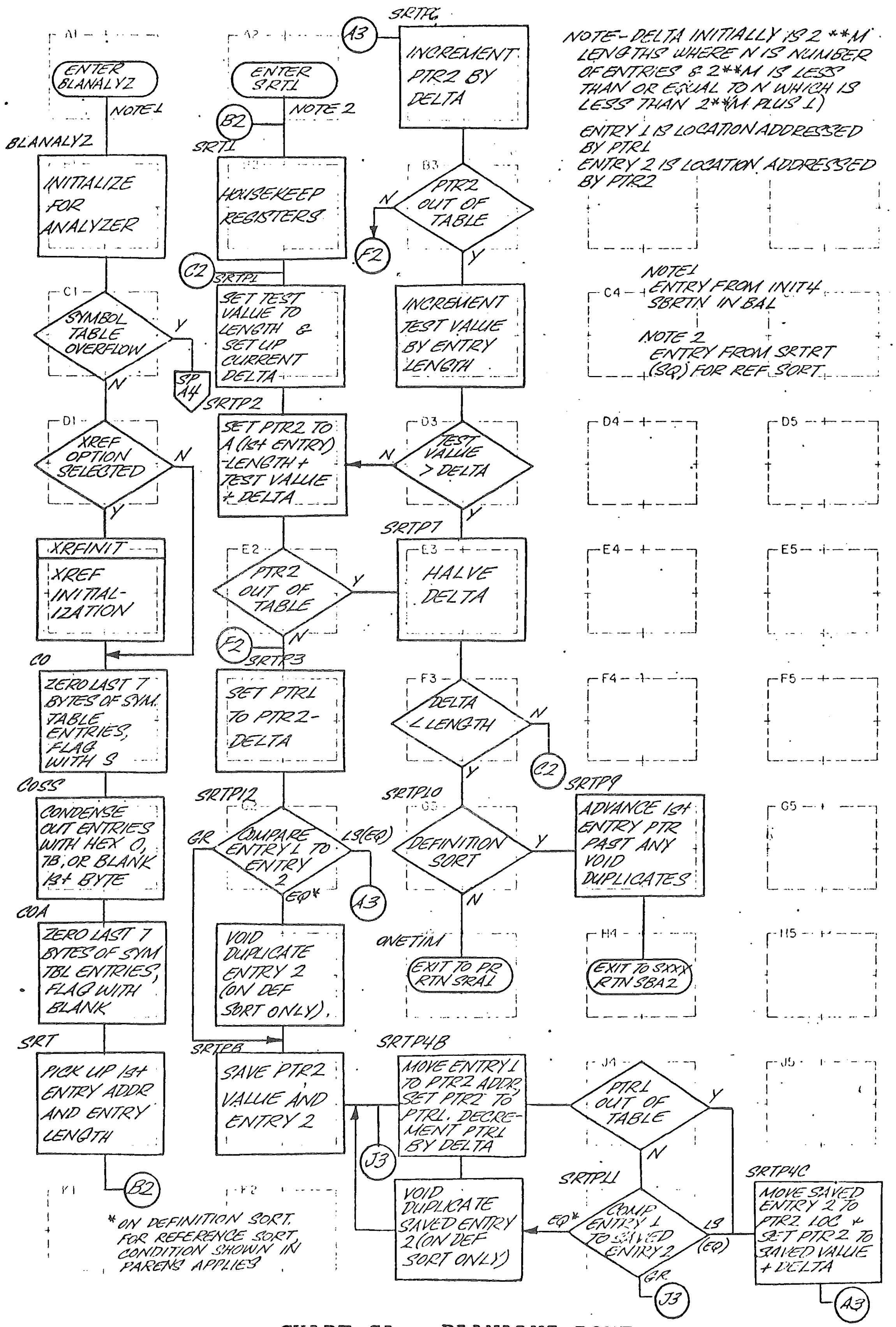
4.20.4 XRFTRL Subroutine

FUNCTION: This subroutine (Chart SZ) terminates the XREF portion of the Analyzer.

ENTRY: This routine is entered at XRFTRL from the MASTXX routine when the analyzer has completed its printing.

OPERATION: This routine calls XRFPCH to punch the last reference card. It then builds the XRF trailer card and calls XRFPCH to punch that card.

EXIT: Exit is made to the calling point in the MASTXX routine.



NOTE- DELTA INITIALLY IS 2 **M LENGTHS WHERE N IS NUMBER OF ENTRIES & 2**M IS LESS THAN OR EQUAL TO N WHICH IS LESS THAN 2**M PLUS 1)
ENTRY 1 IS LOCATION ADDRESSED BY PTR1
ENTRY 2 IS LOCATION ADDRESSED BY PTR2
NOTE 1 ENTRY FROM INIT4 SBRTN IN BAL
NOTE 2 ENTRY FROM SRTRT (SQ) FOR REF SORT
ENTER BLANALYZ
NOTE 1
BLANALYZ
INITIALIZE FOR ANALYZER
SYMBOL TABLE OVERFLOW
Y
N
SP A4
XREF OPTION SELECTED
N
Y
XRFINIT
XREF INITIAL-IZATION
CO
ZERO LAST 7 BYTES OF SYM. TABLE ENTRIES, FLAG WITH S
COSS
CONDENSE OUT ENTRIES WITH HEX O, 7B, OR BLANK 1st BYTE
COA
ZERO LAST 7 BYTES OF SYM TBL ENTRIES, FLAG WITH BLANK
SRT
PICK UP 1st ENTRY ADDR AND ENTRY LENGTH
B2
ENTER SRT1
NOTE 2
B2
SRT1
HOUSEKEEP REGISTERS
C2
SRTP1
SET TEST VALUE TO LENGTH & SET UP CURRENT DELTA
SRTP2
SET PTR2 TO A (1st ENTRY) -LENGTH + TEST VALUE + DELTA
PTR2 OUT OF TABLE
Y
N
F2
SRTP3
SET PTR1 TO PTR2-DELTA
SRTP12
COMPARE ENTRY 1 TO ENTRY 2
GR
LS(EQ)
EQ*
A3
VOID DUPLICATE ENTRY 2 (ON DEF SORT ONLY)
SRTP8
SAVE PTR2 VALUE AND ENTRY 2
A3
SRTP6
INCREMENT PTR2 BY DELTA
PTR2 OUT OF TABLE
N
F2
Y
INCREMENT TEST VALUE BY ENTRY LENGTH
TEST VALUE > DELTA
N
Y
SRTP7
HALVE DELTA
DELTA < LENGTH
N
C2
Y
SRTP10
DEFINITION SORT
Y
N
ONETIM
EXIT TO PR RTN SRA1
SRTP9
ADVANCE 1st ENTRY PTR PAST ANY VOID DUPLICATES
EXIT TO SXXX RTN SBA2
SRTP4B
MOVE ENTRY 1 TO PTR2 ADDR. SET PTR2 TO PTR1. DECRE-MENT PTR1 BY DELTA
J3
PTR1 OUT OF TABLE
Y
N
SRTP11
COMP ENTRY 1 TO SAVED ENTRY 2
EQ*
LS (EQ)
GR
J3
VOID DUPLICATE SAVED ENTRY 2 (ON DEF SORT ONLY)
SRTP4C
MOVE SAVED ENTRY 2 TO PTR2 LOC + SET PTR2 TO SAVED VALUE + DELTA
A3
*ON DEFINITION SORT. FOR REFERENCE SORT, CONDITION SHOWN IN PARENS APPLIES

CHART SA. BLANALYZ ROUTINE

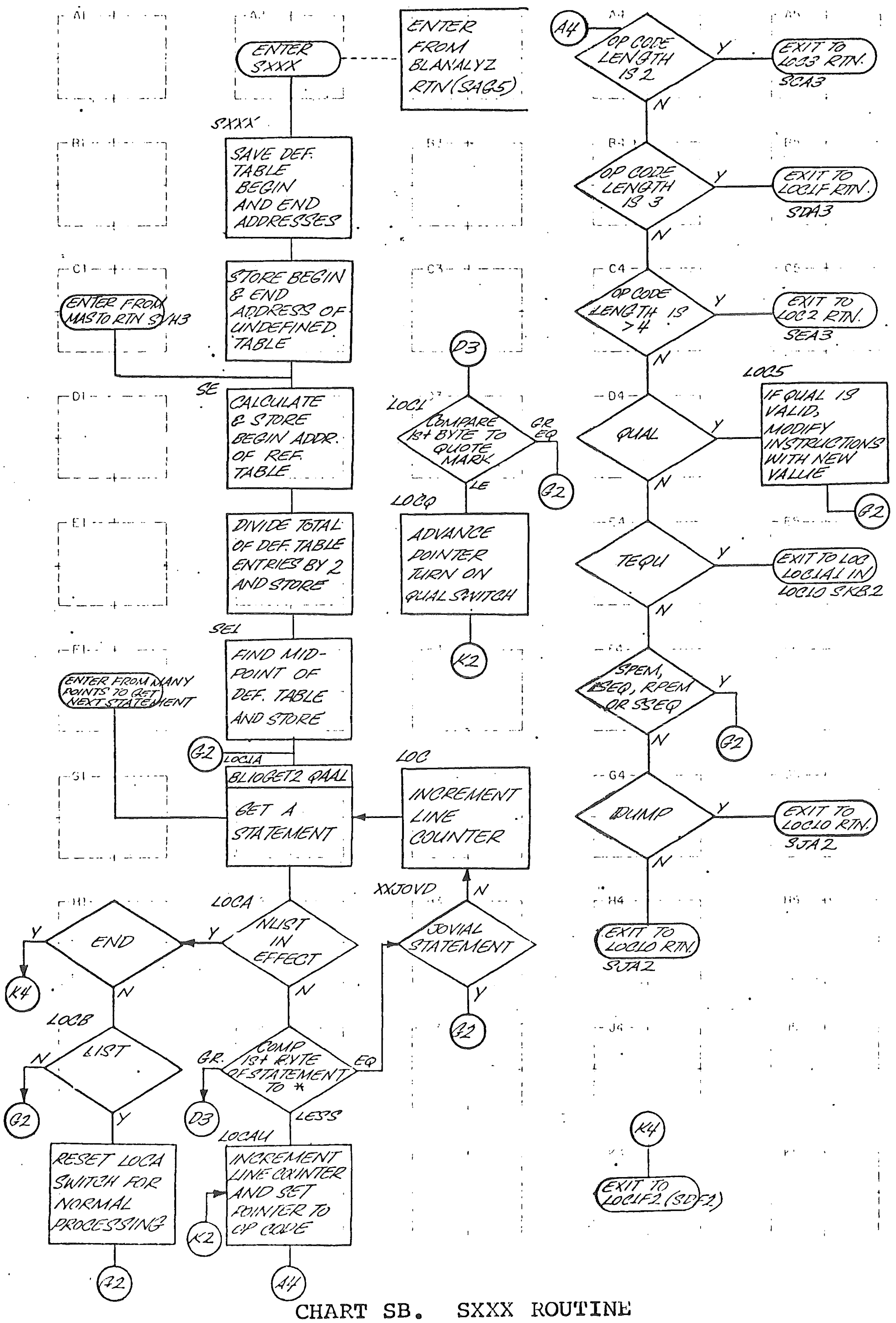

ENTER SXXX
ENTER FROM BLANALYZ RTN (SAG5)
SXXX
SAVE DEF. TABLE BEGIN AND END ADDRESSES
STORE BEGIN & END ADDRESS OF UNDEFINED TABLE
ENTER FROM MASTO RTN SYH3
SE
CALCULATE & STORE BEGIN ADDR. OF REF. TABLE
DIVIDE TOTAL OF DEF. TABLE ENTRIES BY 2 AND STORE
SEL
FIND MID-POINT OF DEF. TABLE AND STORE
ENTER FROM MANY POINTS TO GET NEXT STATEMENT
G2
LOCIA
BLIOGET2 QAAL
GET A STATEMENT
LOC
INCREMENT LINE COUNTER
LOCA
NLIST IN EFFECT
Y
N
END
Y
K4
N
LOCB
LIST
N
G2
Y
RESET LOCA SWITCH FOR NORMAL PROCESSING
G2
COMP 1st BYTE OF STATEMENT TO *
GR.
EQ
LESS
D3
LOCAU
INCREMENT LINE COUNTER AND SET POINTER TO OP CODE
K2
A4
XXJOVD
JOVIAL STATEMENT
N
Y
G2
D3
LOCL
COMPARE 1st BYTE TO QUOTE MARK
GR EQ
G2
LE
LOCQ
ADVANCE POINTER TURN ON QUAL SWITCH
K2
A4
OP CODE LENGTH IS 2
Y
EXIT TO LOC3 RTN. SCA3
N
OP CODE LENGTH IS 3
Y
EXIT TO LOC1F RTN. SDA3
N
OP CODE LENGTH IS >4
Y
EXIT TO LOC2 RTN. SEA3
N
LOC5
QUAL
Y
IF QUAL IS VALID, MODIFY INSTRUCTIONS WITH NEW VALUE
G2
N
TEQU
Y
EXIT TO LOC LOC1A1 IN LOC10 SKB2
N
SPEM, SEQ, RPEM OR SSEQ
Y
G2
N
DUMP
Y
EXIT TO LOC10 RTN. SJA2
N
EXIT TO LOC10 RTN. SJA2
K4
EXIT TO LOC1F2 (SDF2)

CHART SB. SXXX ROUTINE

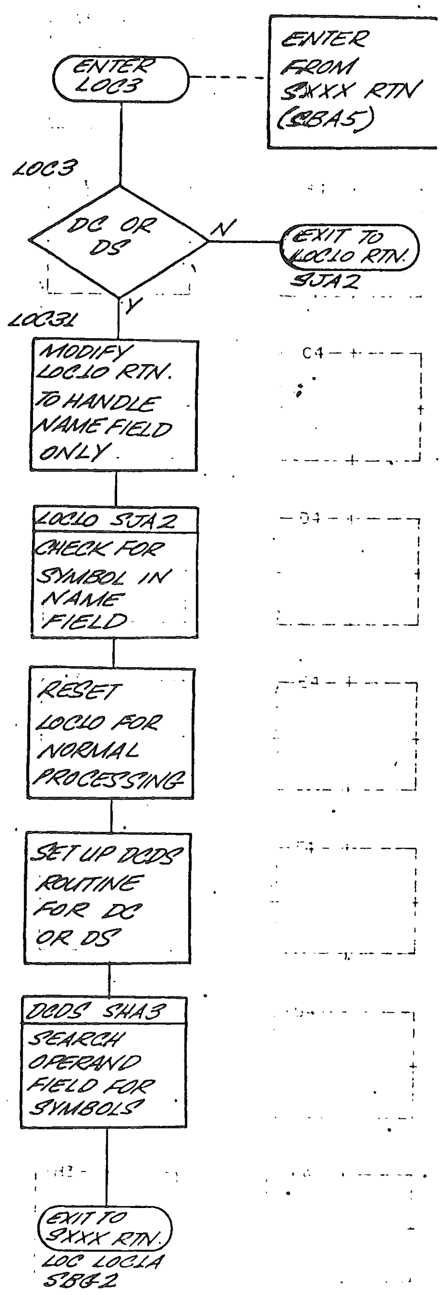

ENTER LOC3
ENTER FROM $XXX RTN ($BA5)
LOC3
DC OR DS
N
EXIT TO LOC10 RTN. $JA2
Y
LOC31
MODIFY LOC10 RTN. TO HANDLE NAME FIELD ONLY
LOC10 $JA2
CHECK FOR SYMBOL IN NAME FIELD
RESET LOC10 FOR NORMAL PROCESSING
SET UP DCDS ROUTINE FOR DC OR DS
DCDS $HA3
SEARCH OPERAND FIELD FOR SYMBOLS
EXIT TO $XXX RTN. LOC LOC1A $BG2

CHART SC. LOC3 ROUTINE

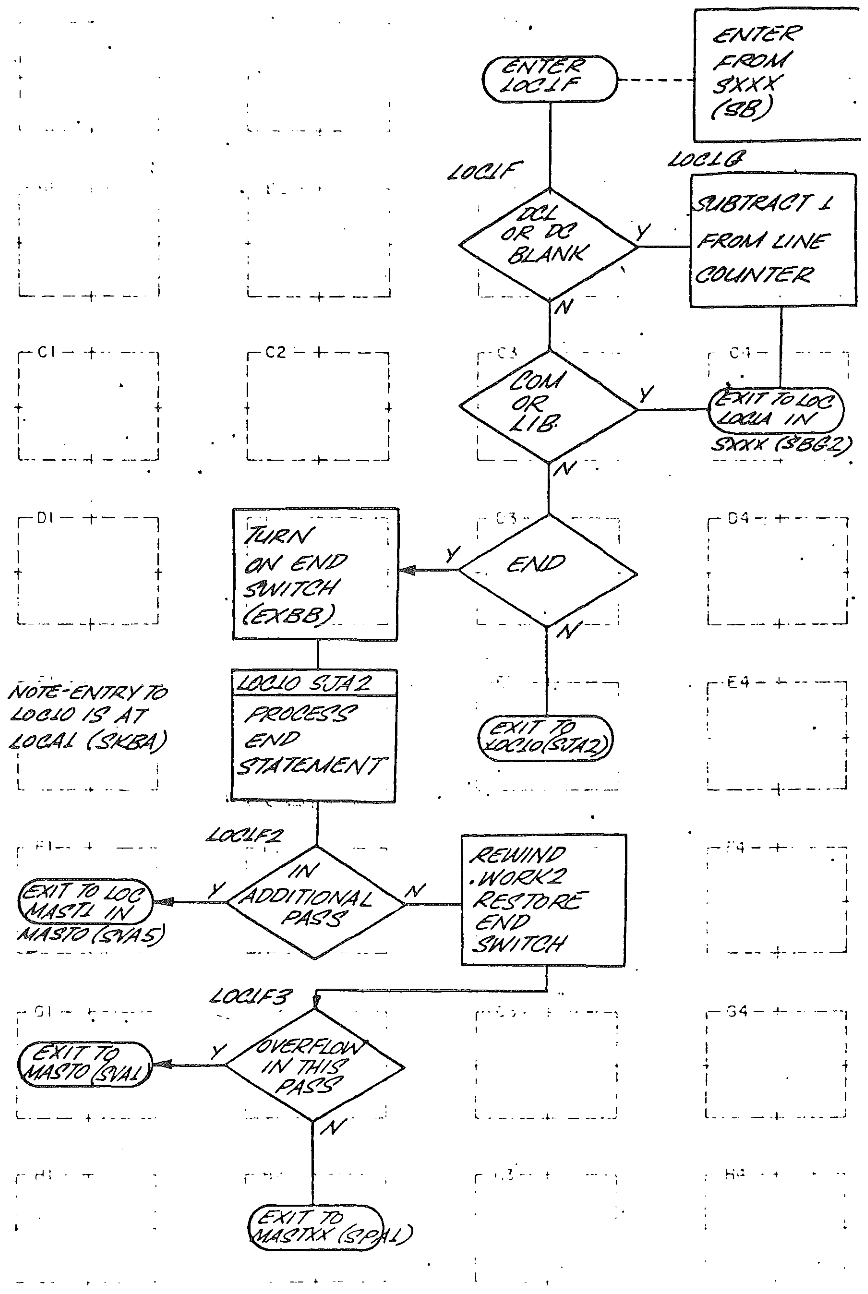


CHART SD. LOC1F ROUTINE

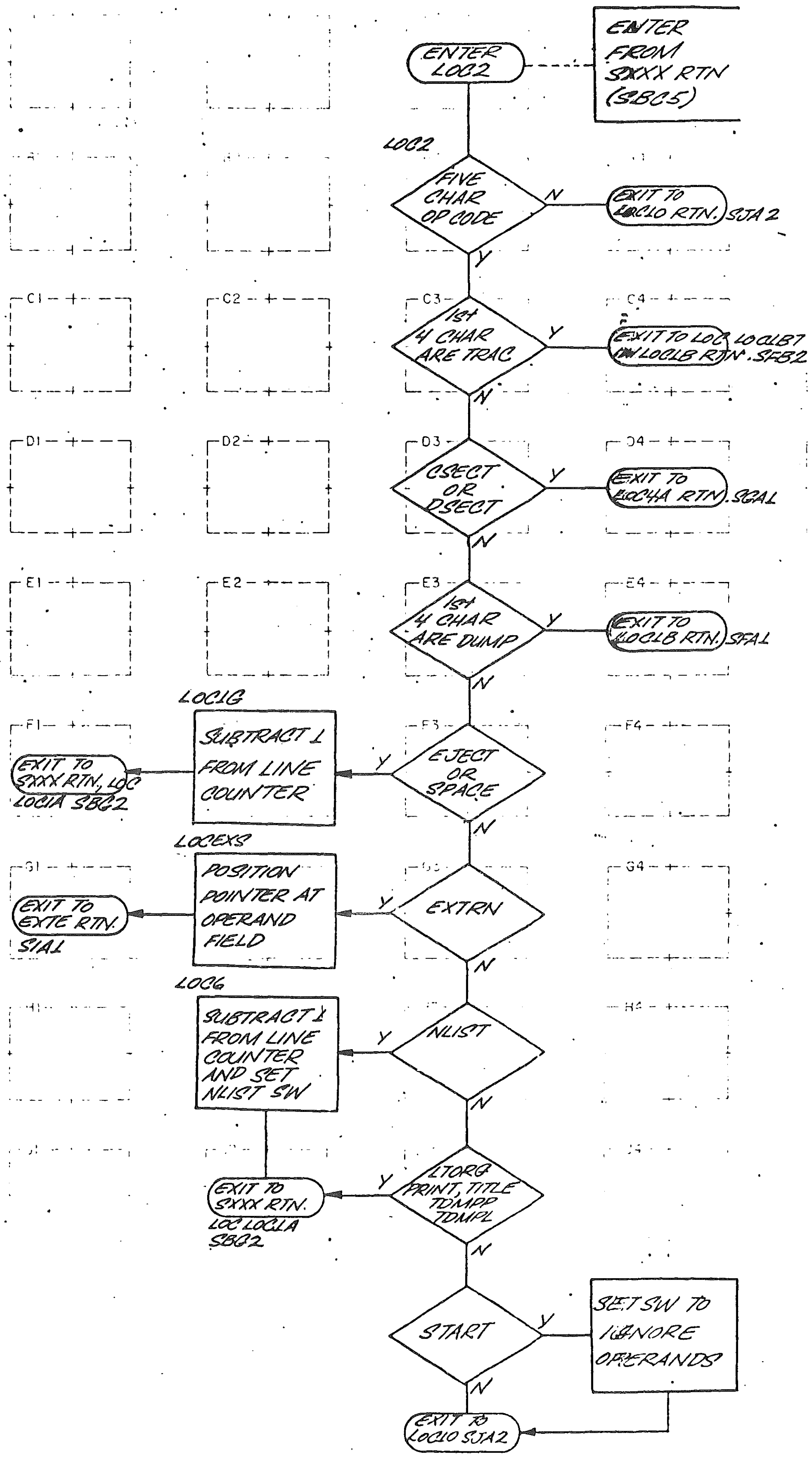

ENTER LOC2
ENTER FROM SXXX RTN (SBC5)
LOC2
FIVE CHAR OP CODE
N
EXIT TO LOC10 RTN. SJA2
Y
1st 4 CHAR ARE TRAC
EXIT TO LOC LOCLB7 ON LOCLB RTN. SFB2
CSECT OR DSECT
EXIT TO LOC4A RTN. SGA1
1st 4 CHAR ARE DUMP
EXIT TO LOCLB RTN. SFA1
LOC1G
EJECT OR SPACE
SUBTRACT 1 FROM LINE COUNTER
EXIT TO SXXX RTN, LOC LOC1A SBG2
LOCEXS
EXTRN
POSITION POINTER AT OPERAND FIELD
EXIT TO EXTE RTN. SIA1
LOC6
NLIST
SUBTRACT 1 FROM LINE COUNTER AND SET NLIST SW
LTORG PRINT, TITLE TOMPP TOMPL
EXIT TO SXXX RTN. LOC LOC1A SBG2
START
SET SW TO IGNORE OPERANDS
EXIT TO LOC10 SJA2
C1
C2
C3
C4
D1
D2
D3
D4
E1
E2
E3
E4
F1
F3
F4
G1
G4
H4

CHART SE. LOC2 ROUTINE

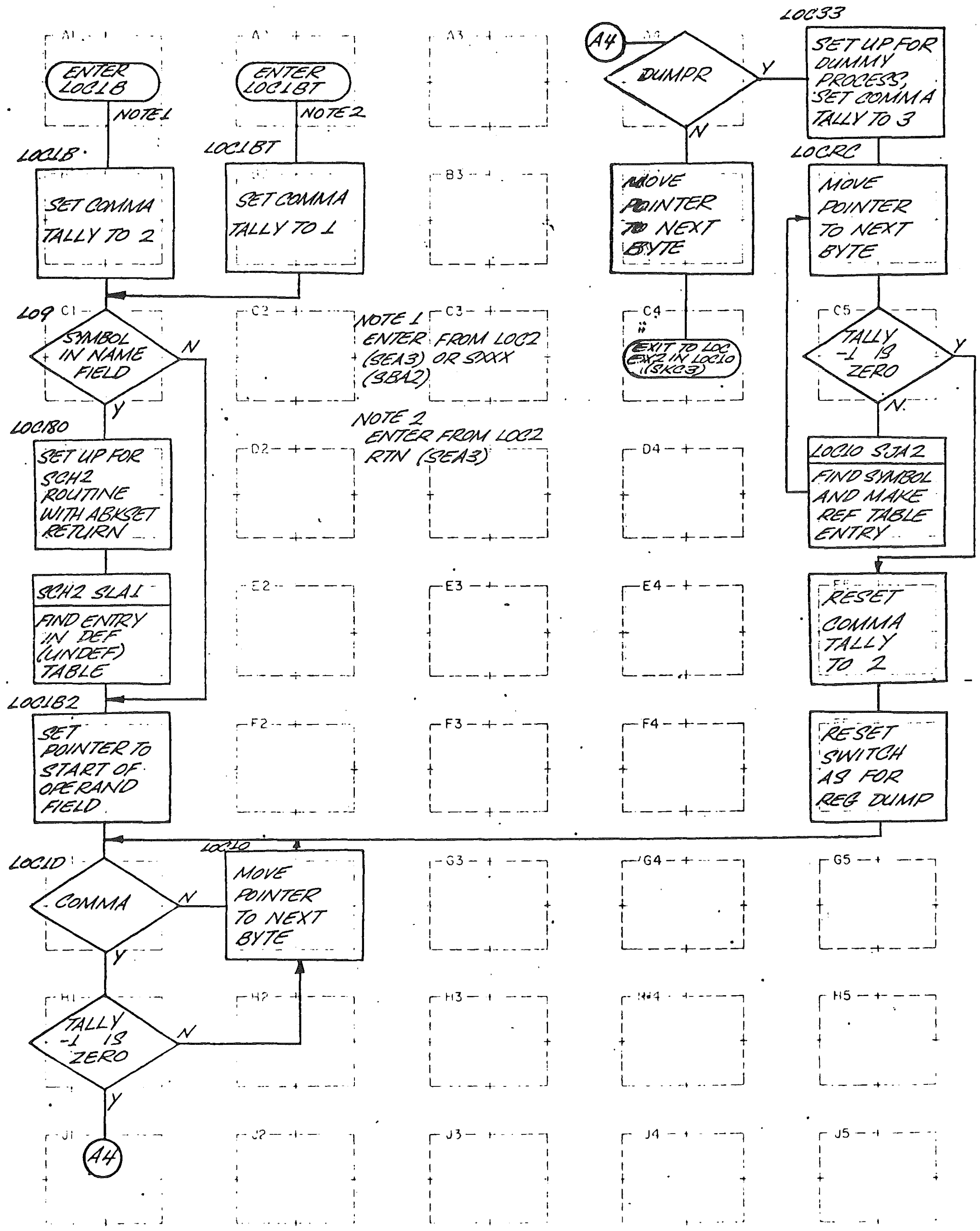
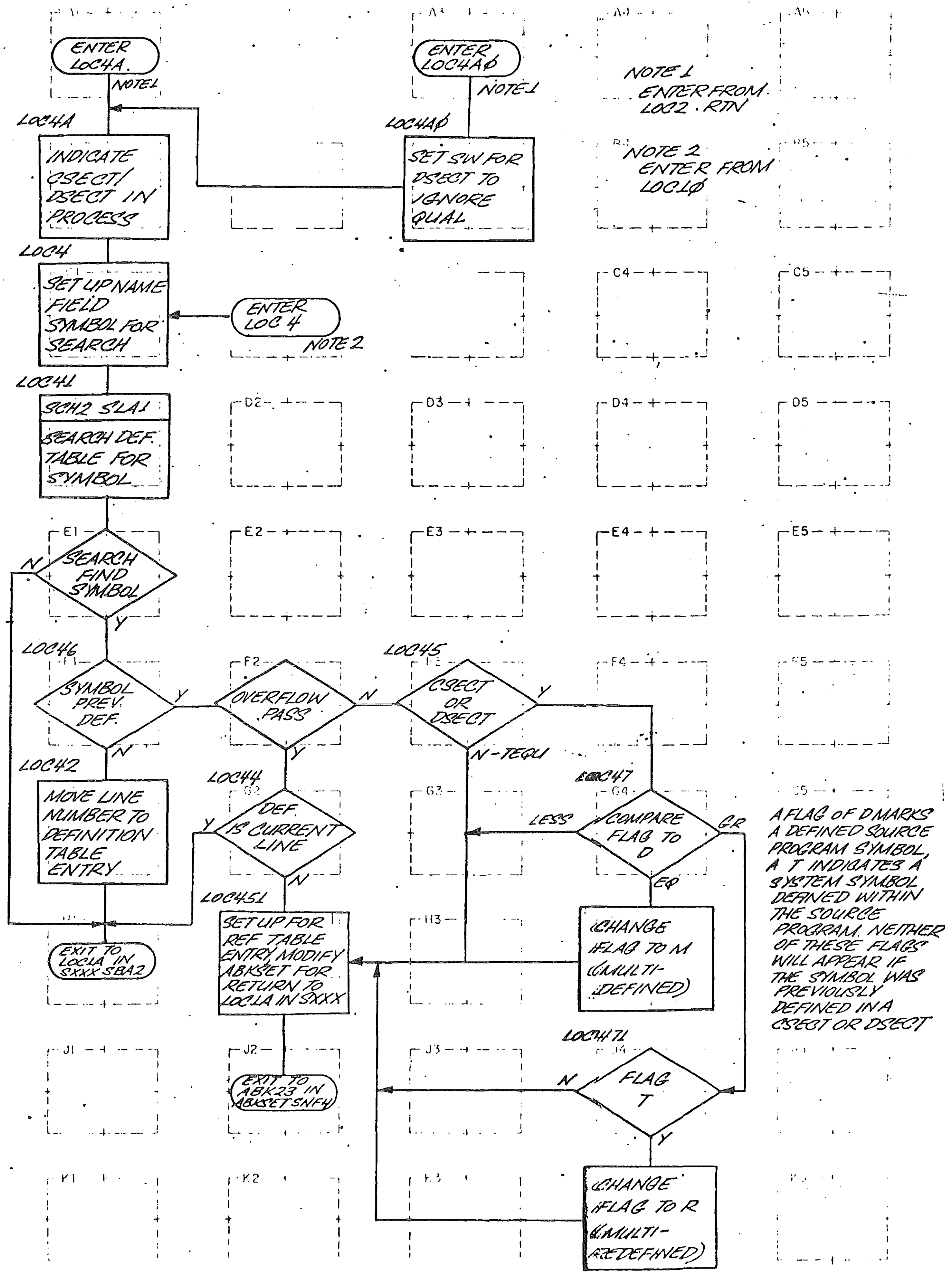

ENTER LOC1B
NOTE 1
LOC1B
SET COMMA TALLY TO 2
ENTER LOC1BT
NOTE 2
LOC1BT
SET COMMA TALLY TO 1
LOC9
SYMBOL IN NAME FIELD
N
Y
LOC180
SET UP FOR SCH2 ROUTINE WITH ABKSET RETURN
SCH2 SLA1
FIND ENTRY IN DEF (UNDEF) TABLE
LOC182
SET POINTER TO START OF OPERAND FIELD
LOC1D
COMMA
N
Y
LOC1D
MOVE POINTER TO NEXT BYTE
TALLY -1 IS ZERO
N
Y
A4
NOTE 1 ENTER FROM LOC2 (SEA3) OR SXXX (SBA2)
NOTE 2 ENTER FROM LOC2 RTN (SEA3)
A4
DUMPR
Y
N
MOVE POINTER TO NEXT BYTE
EXIT TO LOC EX2 IN LOC10 (SKC3)
LOC33
SET UP FOR DUMMY PROCESS, SET COMMA TALLY TO 3
LOCRC
MOVE POINTER TO NEXT BYTE
TALLY -1 IS ZERO
Y
N
LOC10 SJA2
FIND SYMBOL AND MAKE REF TABLE ENTRY
RESET COMMA TALLY TO 2
RESET SWITCH AS FOR REG DUMP

CHART SF. LOC1B ROUTINE



ENTER LOC4A.
NOTE1
LOC4A
INDICATE CSECT/DSECT IN PROCESS
LOC4
SET UP NAME FIELD SYMBOL FOR SEARCH
LOC41
SCH2 SLA1
SEARCH DEF. TABLE FOR SYMBOL
E1
SEARCH FIND SYMBOL
N
Y
LOC46
SYMBOL PREV. DEF.
Y
N
LOC42
MOVE LINE NUMBER TO DEFINITION TABLE ENTRY
EXIT TO LOC1A IN SXXX SBA2
ENTER LOC4A∅
NOTE1
LOC4A∅
SET SW FOR DSECT TO IGNORE QUAL
ENTER LOC 4
NOTE 2
F2
OVERFLOW PASS
N
Y
LOC44
DEF. IS CURRENT LINE
Y
N
LOC451
SET UP FOR REF TABLE ENTRY, MODIFY ABKSET FOR RETURN TO LOC1A IN SXXX
J2
EXIT TO ABK23 IN ABKSET SNF4
LOC45
CSECT OR DSECT
Y
N-TEQU
LOC47
COMPARE FLAG TO D
LESS
GR
EQ
CHANGE FLAG TO M (MULTI-DEFINED)
LOC471
FLAG T
N
Y
CHANGE FLAG TO R (MULTI-REDEFINED)
NOTE 1 ENTER FROM LOC2 · RTN
NOTE 2 ENTER FROM LOC1∅
A FLAG OF D MARKS A DEFINED SOURCE PROGRAM SYMBOL, A T INDICATES A SYSTEM SYMBOL DEFINED WITHIN THE SOURCE PROGRAM. NEITHER OF THESE FLAGS WILL APPEAR IF THE SYMBOL WAS PREVIOUSLY DEFINED IN A CSECT OR DSECT

CHART SG. LOC4A ROUTINE

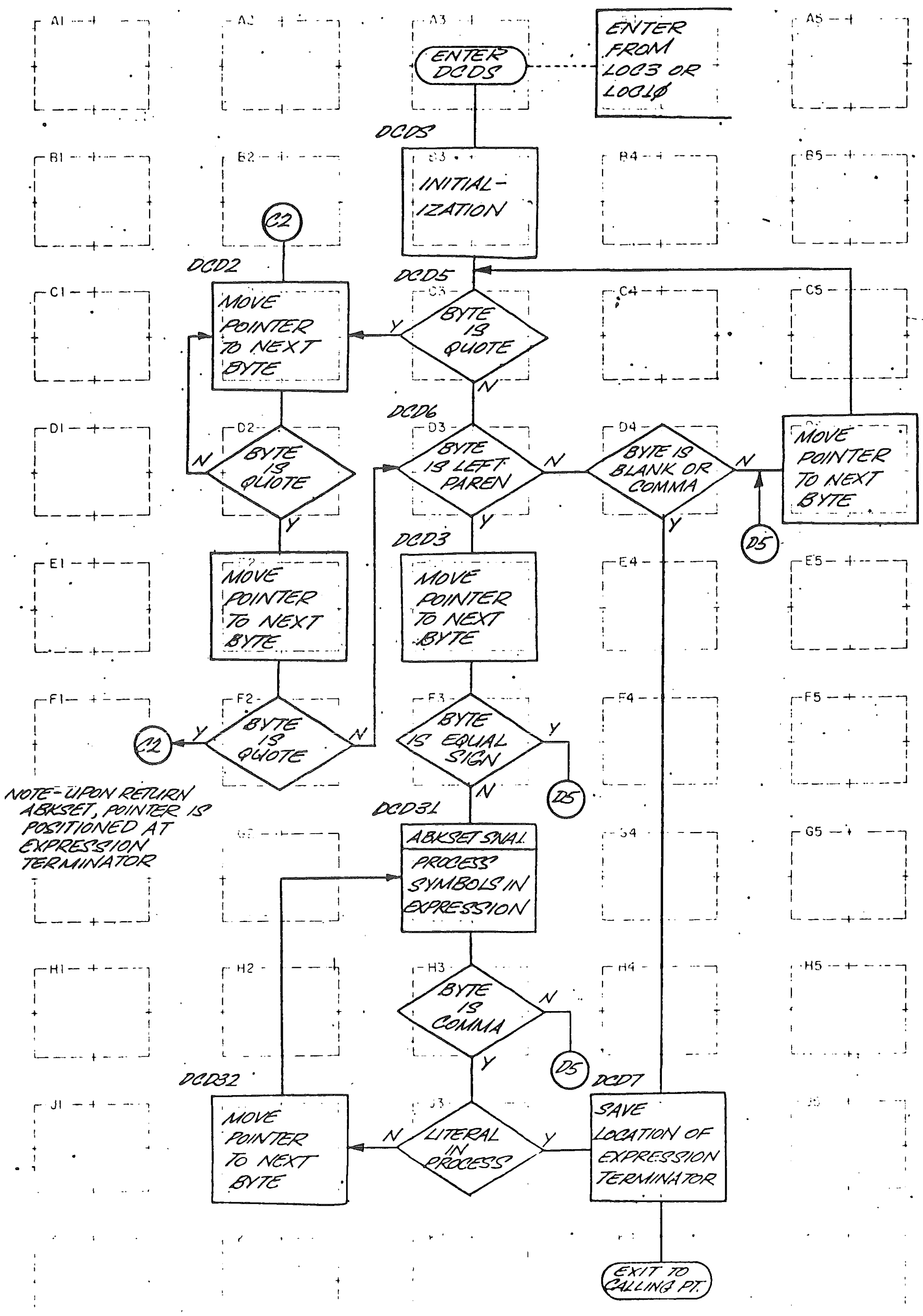


CHART SH. DCDS ROUTINE

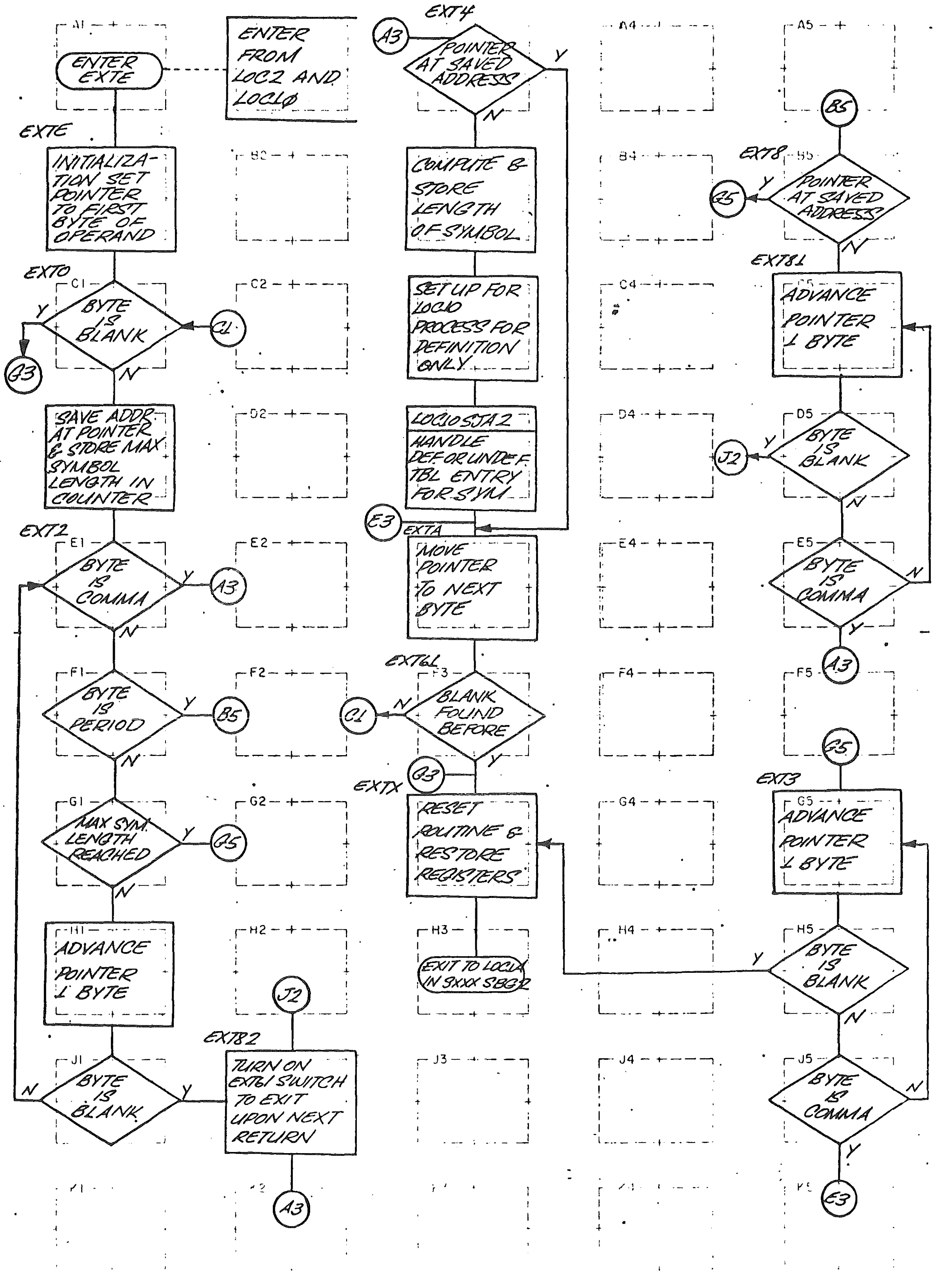
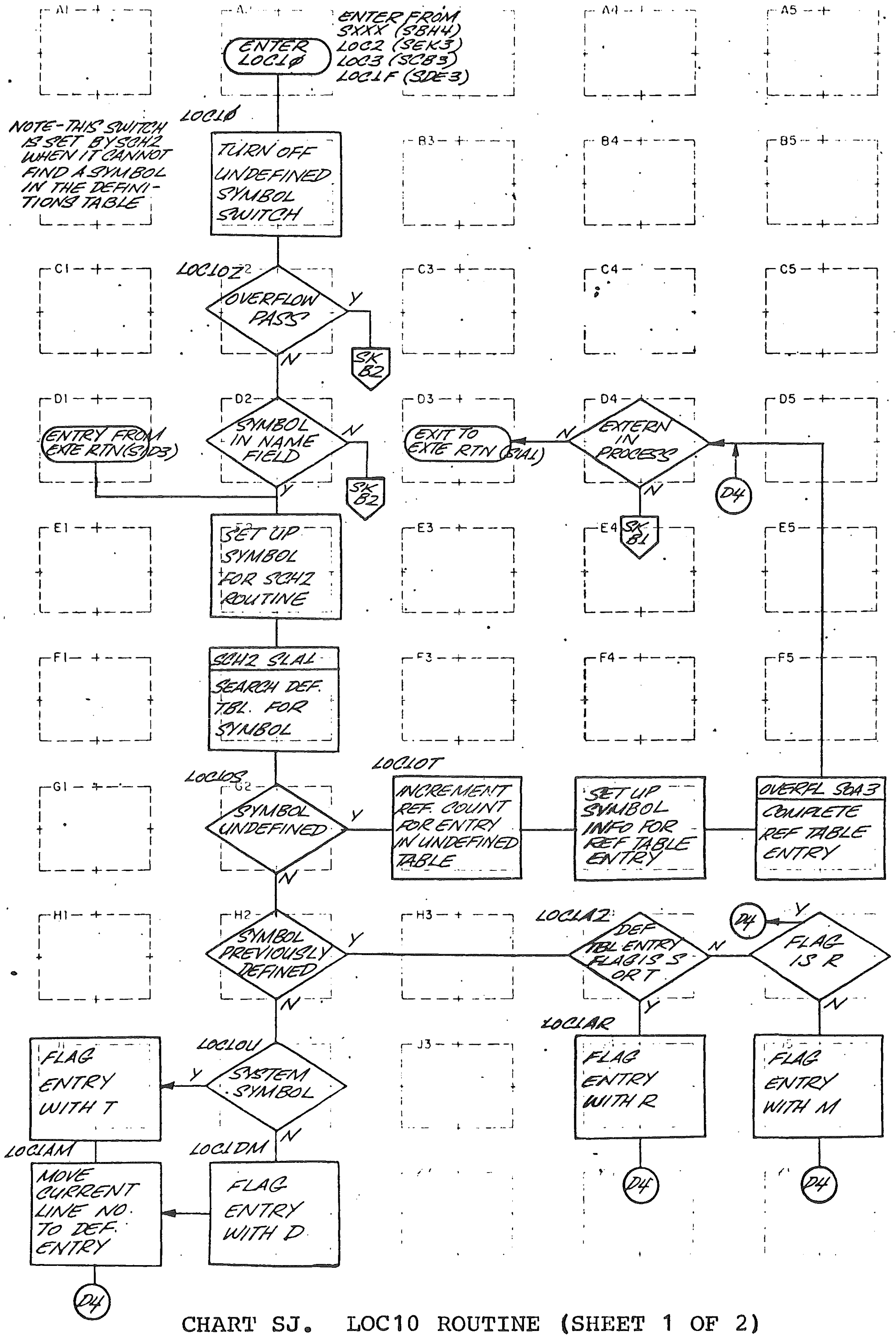

ENTER EXTE
ENTER FROM LOC2 AND LOCLØ
EXTE
INITIALIZATION SET POINTER TO FIRST BYTE OF OPERAND
EXT0
BYTE IS BLANK
SAVE ADDR AT POINTER & STORE MAX SYMBOL LENGTH IN COUNTER
EXT2
BYTE IS COMMA
BYTE IS PERIOD
MAX SYM LENGTH REACHED
ADVANCE POINTER 1 BYTE
BYTE IS BLANK
EXT82
TURN ON EXT61 SWITCH TO EXIT UPON NEXT RETURN
EXT4
POINTER AT SAVED ADDRESS
COMPUTE & STORE LENGTH OF SYMBOL
SET UP FOR LOC10 PROCESS FOR DEFINITION ONLY
LOC10 SJA2
HANDLE DEF OR UNDEF TBL ENTRY FOR SYM
EXTA
MOVE POINTER TO NEXT BYTE
EXT61
BLANK FOUND BEFORE
EXTX
RESET ROUTINE & RESTORE REGISTERS
EXIT TO LOCLA IN 9XXX SBG2
EXT8
POINTER AT SAVED ADDRESS?
EXT81
ADVANCE POINTER 1 BYTE
BYTE IS BLANK
BYTE IS COMMA
EXT3
ADVANCE POINTER 1 BYTE
BYTE IS BLANK
BYTE IS COMMA

CHART SI. EXTE ROUTINE



ENTER FROM SXXX (SBH4) LOC2 (SEK3) LOC3 (SC83) LOC1F (SDE3)
ENTER LOC1Ø
NOTE-THIS SWITCH IS SET BY SCH2 WHEN IT CANNOT FIND A SYMBOL IN THE DEFINITIONS TABLE
LOC1Ø
TURN OFF UNDEFINED SYMBOL SWITCH
LOC1OZ
OVERFLOW PASS?
Y
N
SK B2
ENTRY FROM EXTE RTN(SID3)
SYMBOL IN NAME FIELD
N
Y
SK B2
EXIT TO EXTE RTN (SIA1)
N
EXTERN IN PROCESS
N
SK B1
D4
SET UP SYMBOL FOR SCH2 ROUTINE
SCH2 SLA1
SEARCH DEF. TBL. FOR SYMBOL
LOC1OS
SYMBOL UNDEFINED
Y
N
LOC1OT
INCREMENT REF. COUNT FOR ENTRY IN UNDEFINED TABLE
SET UP SYMBOL INFO FOR REF TABLE ENTRY
OVERFL SDA3
COMPLETE REF TABLE ENTRY
SYMBOL PREVIOUSLY DEFINED
Y
N
LOC1A2
DEF TBL ENTRY FLAG IS S OR T
N
Y
D4
Y
FLAG IS R
N
LOC1OU
SYSTEM SYMBOL
Y
N
FLAG ENTRY WITH T
LOC1AM
MOVE CURRENT LINE NO. TO DEF. ENTRY
LOC1DM
FLAG ENTRY WITH D
D4
LOC1AR
FLAG ENTRY WITH R
D4
FLAG ENTRY WITH M
D4

CHART SJ. LOC10 ROUTINE (SHEET 1 OF 2)

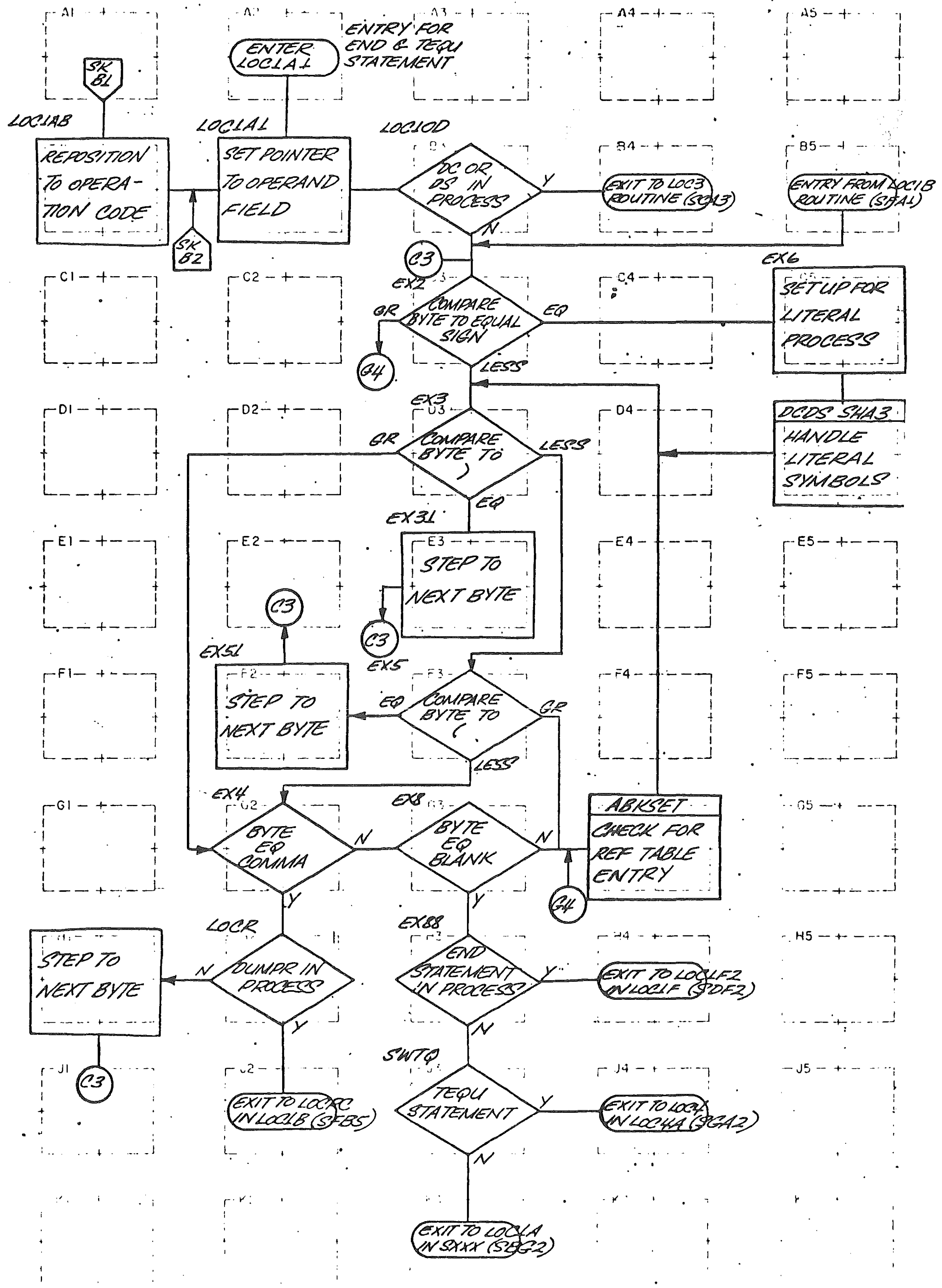


CHART SK. LOC10 ROUTINE (SHEET 2 OF 2)

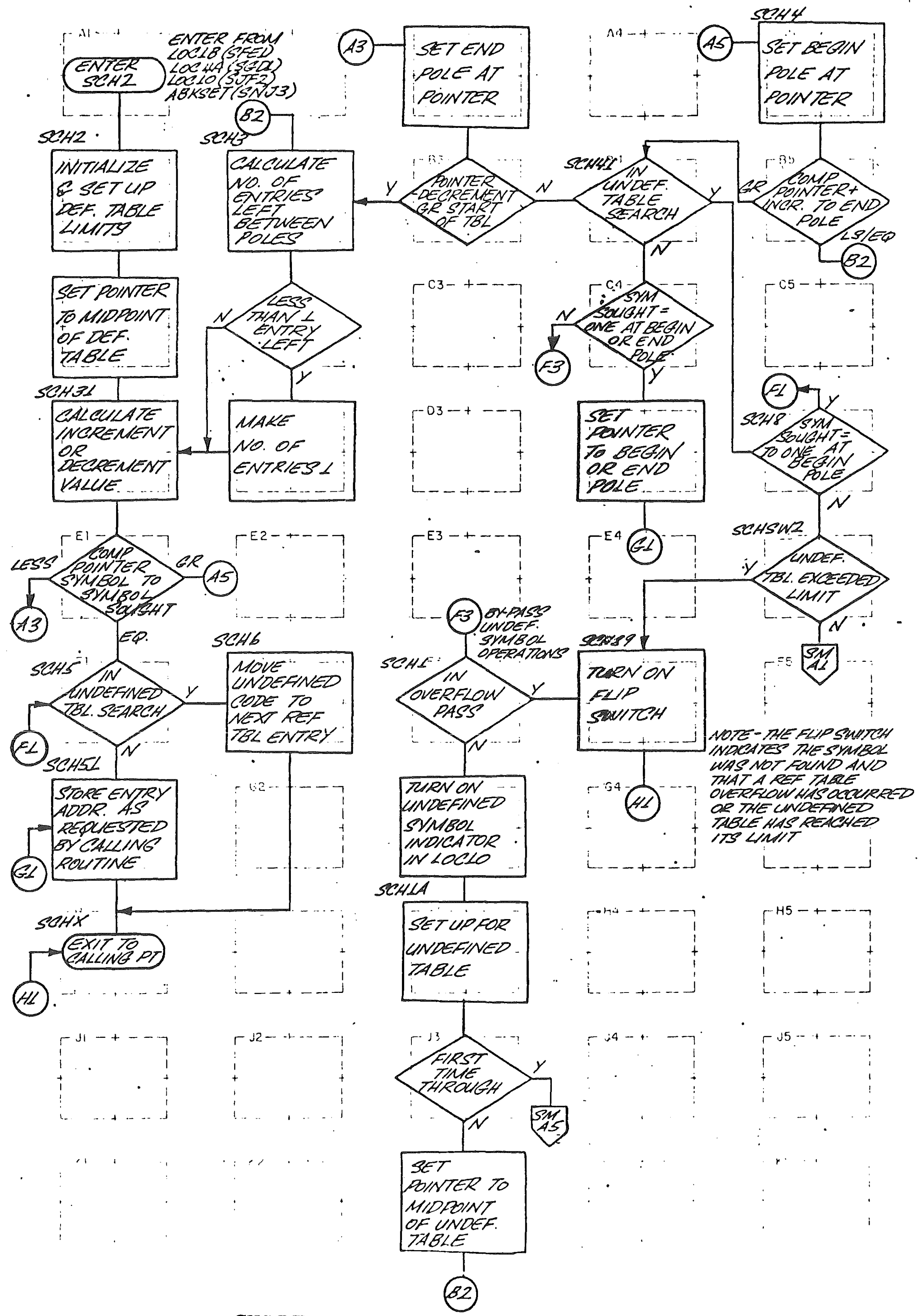

ENTER FROM LOC18 (SFED) LOC4A (SGDL) LOC10 (SJF2) ABKSET (SNJ3)
ENTER SCH2
SCH2
INITIALIZE & SET UP DEF. TABLE LIMITS
SET POINTER TO MIDPOINT OF DEF. TABLE
SCH31
CALCULATE INCREMENT OR DECREMENT VALUE
COMP POINTER SYMBOL TO SYMBOL SOUGHT
LESS
GR
EQ.
SCH5
IN UNDEFINED TBL. SEARCH
SCH5L
STORE ENTRY ADDR. AS REQUESTED BY CALLING ROUTINE
SCHX
EXIT TO CALLING PT
SCH3
CALCULATE NO. OF ENTRIES LEFT BETWEEN POLES
LESS THAN 1 ENTRY LEFT
MAKE NO. OF ENTRIES 1
SCH6
MOVE UNDEFINED CODE TO NEXT REF TBL ENTRY
SET END POLE AT POINTER
POINTER DECREMENT GR START OF TBL
SCH4L
IN UNDEF. TABLE SEARCH
SYM SOUGHT = ONE AT BEGIN OR END POLE
SET POINTER TO BEGIN OR END POLE
BY-PASS UNDEF. SYMBOL OPERATIONS
SCHLE
IN OVERFLOW PASS
TURN ON UNDEFINED SYMBOL INDICATOR IN LOC10
SCHLA
SET UP FOR UNDEFINED TABLE
FIRST TIME THROUGH
SET POINTER TO MIDPOINT OF UNDEF. TABLE
SCH89
TURN ON FLIP SWITCH
NOTE - THE FLIP SWITCH INDICATES THE SYMBOL WAS NOT FOUND AND THAT A REF TABLE OVERFLOW HAS OCCURRED OR THE UNDEFINED TABLE HAS REACHED ITS LIMIT
SCH4
SET BEGIN POLE AT POINTER
COMP POINTER + INCR. TO END POLE
LS/EQ
SCH8
SYM SOUGHT = TO ONE AT BEGIN POLE
SCHSW2
UNDEF. TBL. EXCEEDED LIMIT
SM A1
SM A5
Y
N
A3
A5
B2
F3
FL
GL
HL

CHART SL. SCH2 ROUTINE (SHEET 1 OF 2)

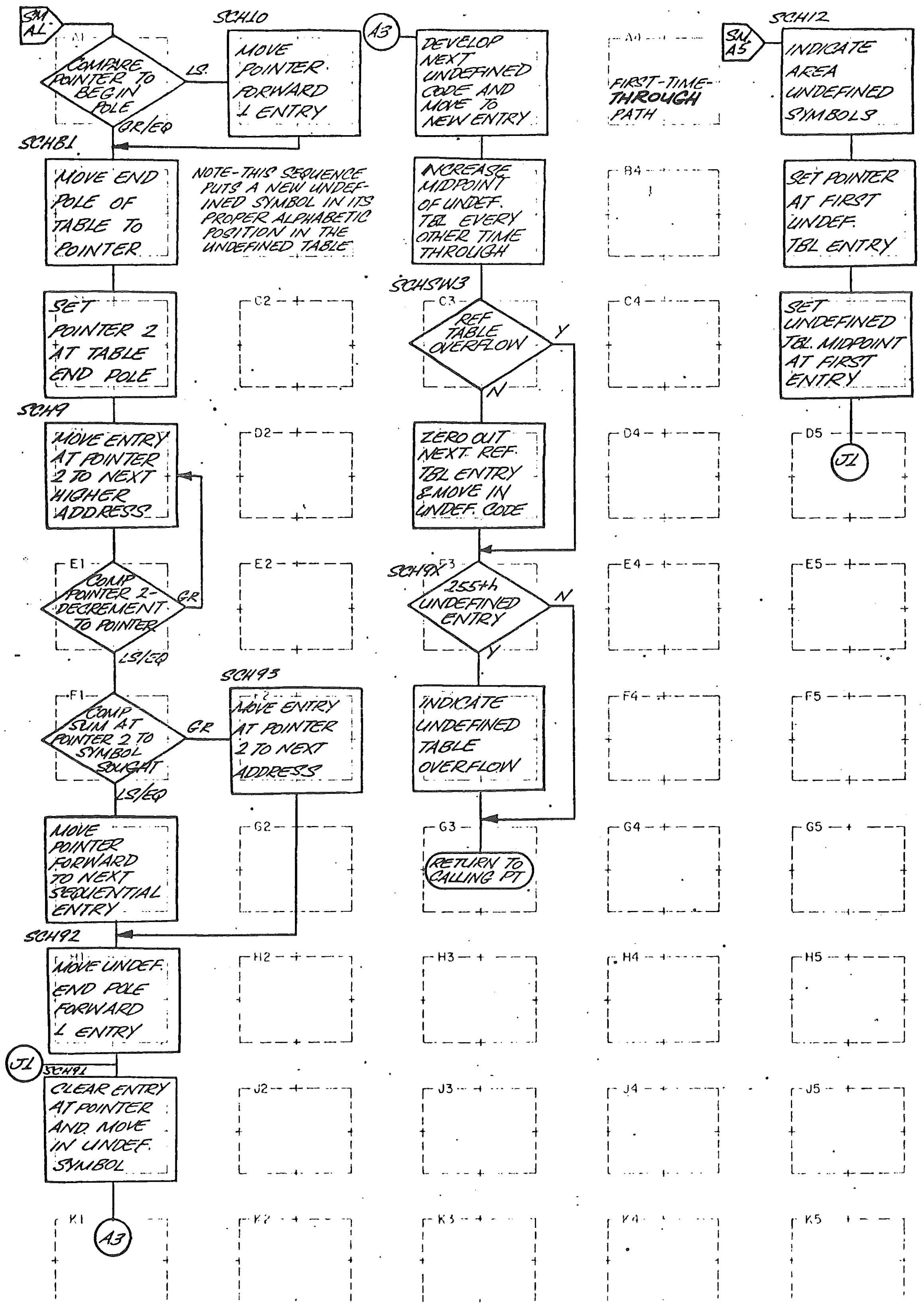
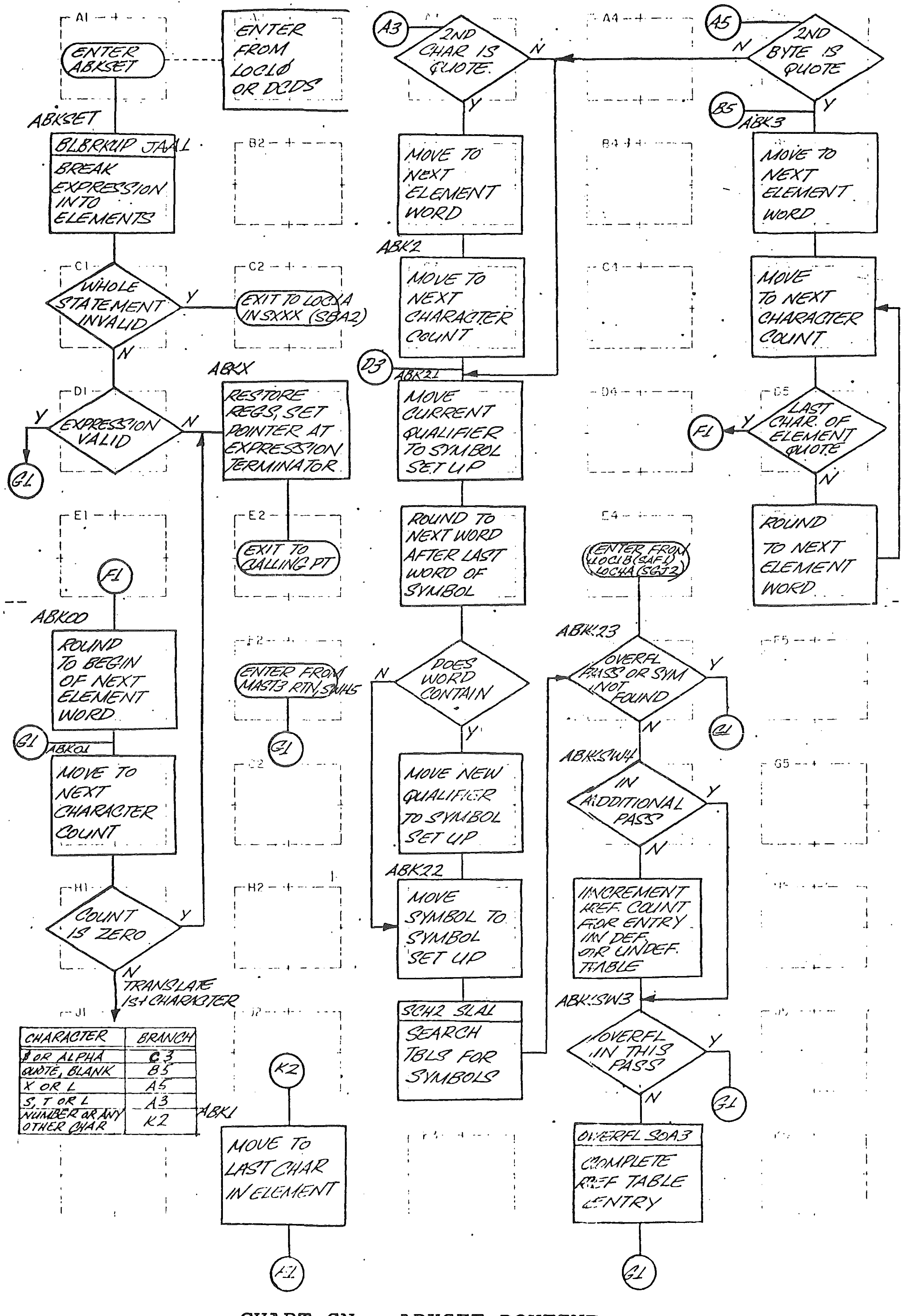

SM AL
COMPARE POINTER TO BEGIN POLE
LS
GR/EQ
SCH10
MOVE POINTER FORWARD 1 ENTRY
SCH81
MOVE END POLE OF TABLE TO POINTER
NOTE-THIS SEQUENCE PUTS A NEW UNDEFINED SYMBOL IN ITS PROPER ALPHABETIC POSITION IN THE UNDEFINED TABLE
SET POINTER 2 AT TABLE END POLE
SCH9
MOVE ENTRY AT POINTER 2 TO NEXT HIGHER ADDRESS
COMP POINTER 2-DECREMENT TO POINTER
GR
LS/EQ
COMP SUM AT POINTER 2 TO SYMBOL SOUGHT
GR
LS/EQ
SCH93
MOVE ENTRY AT POINTER 2 TO NEXT ADDRESS
MOVE POINTER FORWARD TO NEXT SEQUENTIAL ENTRY
SCH92
MOVE UNDEF. END POLE FORWARD 1 ENTRY
J1
SCH91
CLEAR ENTRY AT POINTER AND MOVE IN UNDEF. SYMBOL
A3
A3
DEVELOP NEXT UNDEFINED CODE AND MOVE TO NEW ENTRY
INCREASE MIDPOINT OF UNDEF. TBL EVERY OTHER TIME THROUGH
SCHSW3
REF TABLE OVERFLOW
Y
N
ZERO OUT NEXT REF. TBL ENTRY & MOVE IN UNDEF. CODE
SCH9X
255th UNDEFINED ENTRY
N
Y
INDICATE UNDEFINED TABLE OVERFLOW
RETURN TO CALLING PT
FIRST-TIME-THROUGH PATH
SM A5
SCH12
INDICATE AREA UNDEFINED SYMBOLS
SET POINTER AT FIRST UNDEF. TBL ENTRY
SET UNDEFINED TBL. MIDPOINT AT FIRST ENTRY
J1

CHART SM. SCH2 ROUTINE (SHEET 2 OF 2)


ENTER ABKSET
ENTER FROM LOC1Ø OR DCDS
ABKSET
BLBRKUP JAAL
BREAK EXPRESSION INTO ELEMENTS
WHOLE STATEMENT INVALID
Y
N
EXIT TO LOC1A IN SXXX (SBA2)
ABKX
EXPRESSION VALID
Y
N
G1
RESTORE REGS, SET POINTER AT EXPRESSION TERMINATOR
EXIT TO CALLING PT
F1
ABK00
ROUND TO BEGIN OF NEXT ELEMENT WORD
G1
ABK01
MOVE TO NEXT CHARACTER COUNT
COUNT IS ZERO
Y
N
TRANSLATE 1st CHARACTER
ENTER FROM MAST3 RTN, SWH5
G1
ABK1
K2
MOVE TO LAST CHAR IN ELEMENT
F1
A3
2ND CHAR IS QUOTE
N
Y
MOVE TO NEXT ELEMENT WORD
ABK2
MOVE TO NEXT CHARACTER COUNT
D3
ABK21
MOVE CURRENT QUALIFIER TO SYMBOL SET UP
ROUND TO NEXT WORD AFTER LAST WORD OF SYMBOL
DOES WORD CONTAIN .
N
Y
MOVE NEW QUALIFIER TO SYMBOL SET UP
ABK22
MOVE SYMBOL TO SYMBOL SET UP
SCH2 SLAL
SEARCH TBLS FOR SYMBOLS
ENTER FROM LOC1B (SAFL) LOC4A (SGJ2)
ABK23
OVERFL PASS OR SYM NOT FOUND
Y
N
G1
ABKSW4
IN ADDITIONAL PASS
Y
N
INCREMENT REF. COUNT FOR ENTRY IN DEF. OR UNDEF. TABLE
ABKSW3
OVERFL IN THIS PASS
Y
N
G1
OVERFL SOA3
COMPLETE REF TABLE ENTRY
G1
A5
2ND BYTE IS QUOTE
N
Y
B5
ABK3
MOVE TO NEXT ELEMENT WORD
MOVE TO NEXT CHARACTER COUNT
LAST CHAR. OF ELEMENT QUOTE
Y
N
F1
ROUND TO NEXT ELEMENT WORD

CHARACTER	BRANCH
$ OR ALPHA	C3
QUOTE, BLANK	B5
X OR L	A5
S, T OR L	A3
NUMBER OR ANY OTHER CHAR	K2

CHART SN. ABKSET ROUTINE

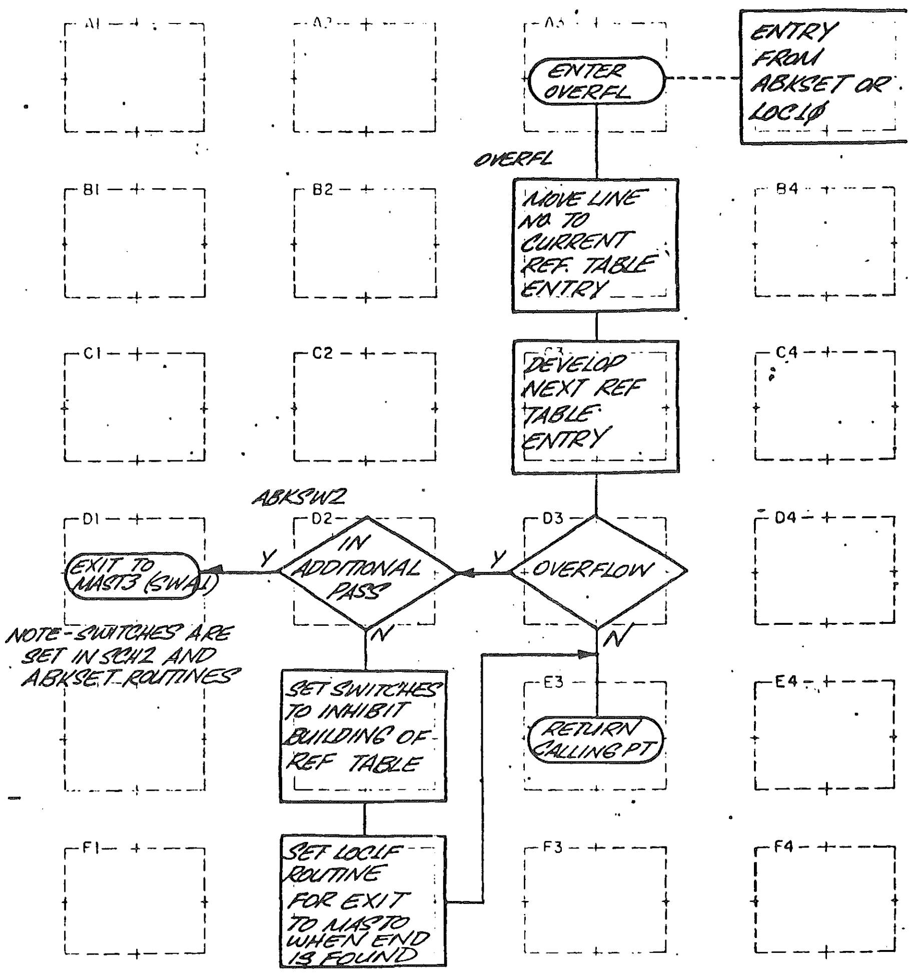

ENTRY FROM ABKSET OR LOC1Ø
ENTER OVERFL
OVERFL
MOVE LINE NO. TO CURRENT REF. TABLE ENTRY
DEVELOP NEXT REF TABLE ENTRY
OVERFLOW
Y
N
ABKSW2
IN ADDITIONAL PASS
EXIT TO MAST3 (SWA1)
NOTE - SWITCHES ARE SET IN SCH2 AND ABKSET ROUTINES
SET SWITCHES TO INHIBIT BUILDING OF REF TABLE
SET LOCLF ROUTINE FOR EXIT TO MASTO WHEN END IS FOUND
RETURN CALLING PT

CHART SO. OVERFL ROUTINE

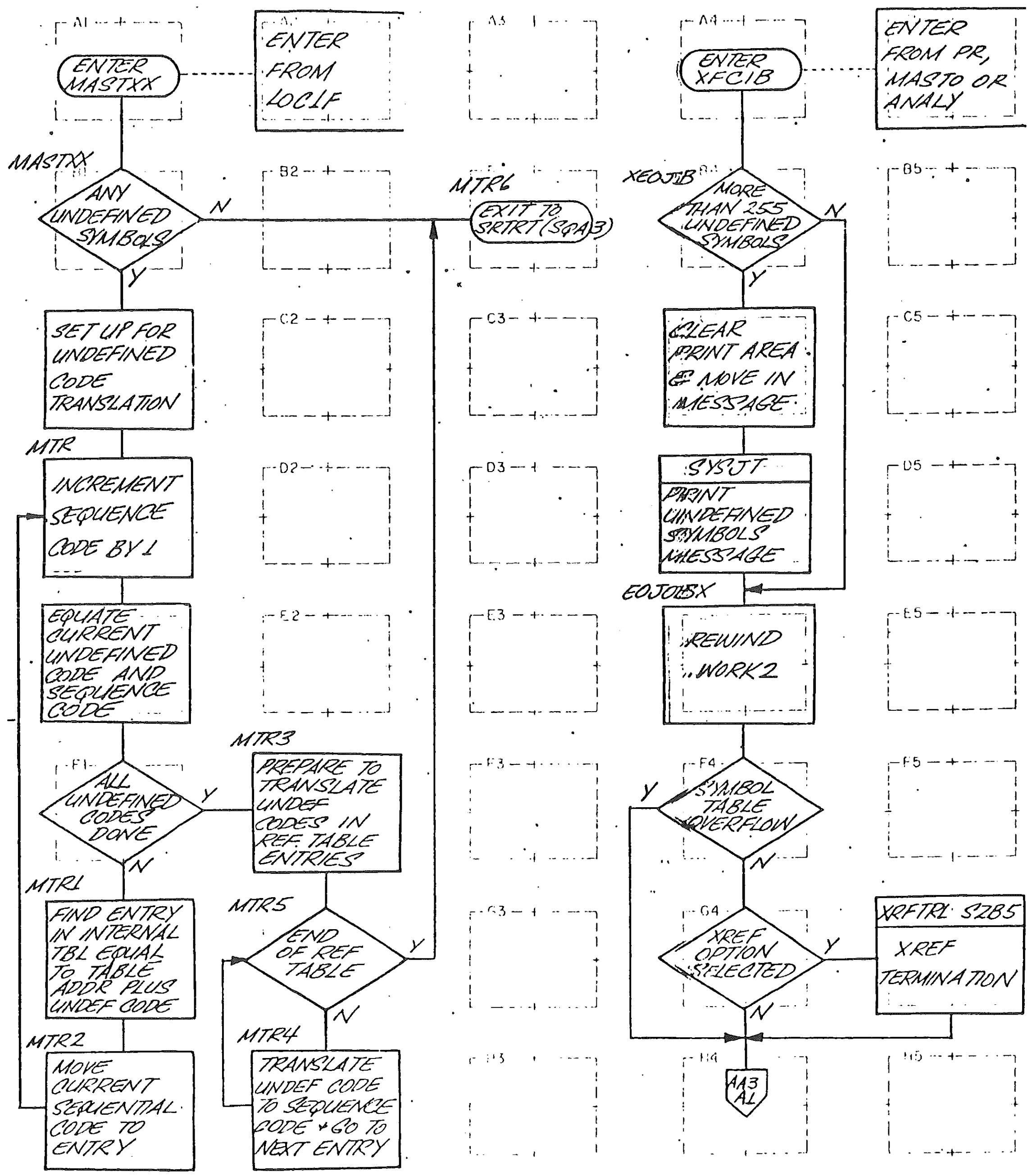

A1
ENTER MASTXX
ENTER FROM LOC1F
A4
ENTER XFC1B
ENTER FROM PR, MASTO OR ANALY
MASTXX
ANY UNDEFINED SYMBOLS
N
Y
B2
MTR6
EXIT TO SRTRT (SPA3)
XEOJIB
MORE THAN 255 UNDEFINED SYMBOLS
N
Y
B5
SET UP FOR UNDEFINED CODE TRANSLATION
C2
C3
CLEAR PRINT AREA & MOVE IN MESSAGE
C5
MTR
INCREMENT SEQUENCE CODE BY 1
D2
D3
SYSJT
PRINT UNDEFINED SYMBOLS MESSAGE
D5
EOJOBX
EQUATE CURRENT UNDEFINED CODE AND SEQUENCE CODE
E2
E3
REWIND WORK2
E5
F1
ALL UNDEFINED CODES DONE
Y
N
MTR3
PREPARE TO TRANSLATE UNDEF CODES IN REF TABLE ENTRIES
F3
F4
SYMBOL TABLE OVERFLOW
Y
N
F5
MTR1
FIND ENTRY IN INTERNAL TBL EQUAL TO TABLE ADDR PLUS UNDEF CODE
MTR5
END OF REF TABLE
Y
N
G3
G4
XREF OPTION SELECTED
Y
N
XRFTRL S2B5
XREF TERMINATION
MTR2
MOVE CURRENT SEQUENTIAL CODE TO ENTRY
MTR4
TRANSLATE UNDEF CODE TO SEQUENCE CODE + GO TO NEXT ENTRY
H3
H4
AA3 A1
H5

CHART SP. MASTXX ROUTINE

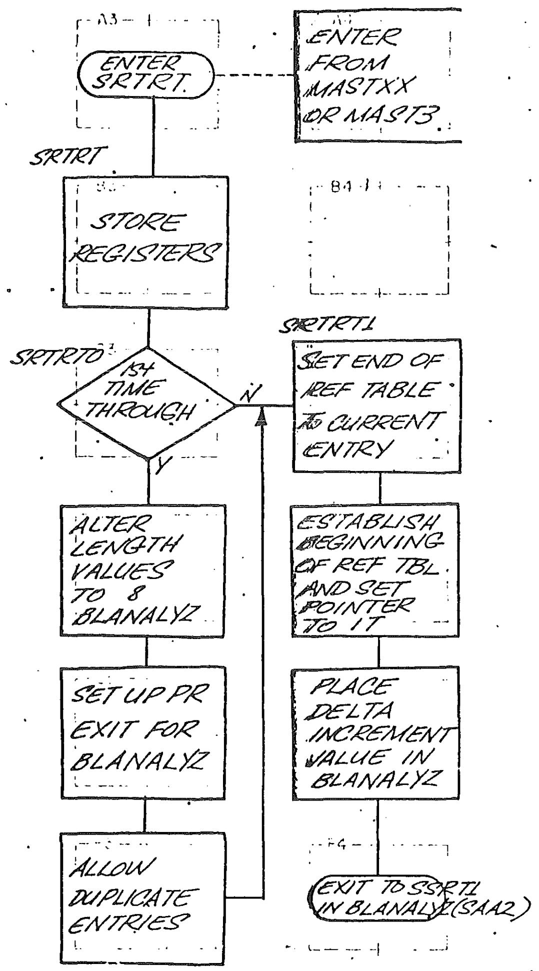

A3
ENTER SRTRT
A4
ENTER FROM MASTXX OR MAST3
SRTRT
STORE REGISTERS
B4
SRTRT1
SRTRT0
1ST TIME THROUGH
N
Y
SET END OF REF TABLE TO CURRENT ENTRY
ALTER LENGTH VALUES TO 8 BLANALYZ
ESTABLISH BEGINNING OF REF TBL AND SET POINTER TO IT
SET UP PR EXIT FOR BLANALYZ
PLACE DELTA INCREMENT VALUE IN BLANALYZ
ALLOW DUPLICATE ENTRIES
F4
EXIT TO SSRT1 IN BLANALYZ (SAA2)

CHART SQ. SRTRT ROUTINE

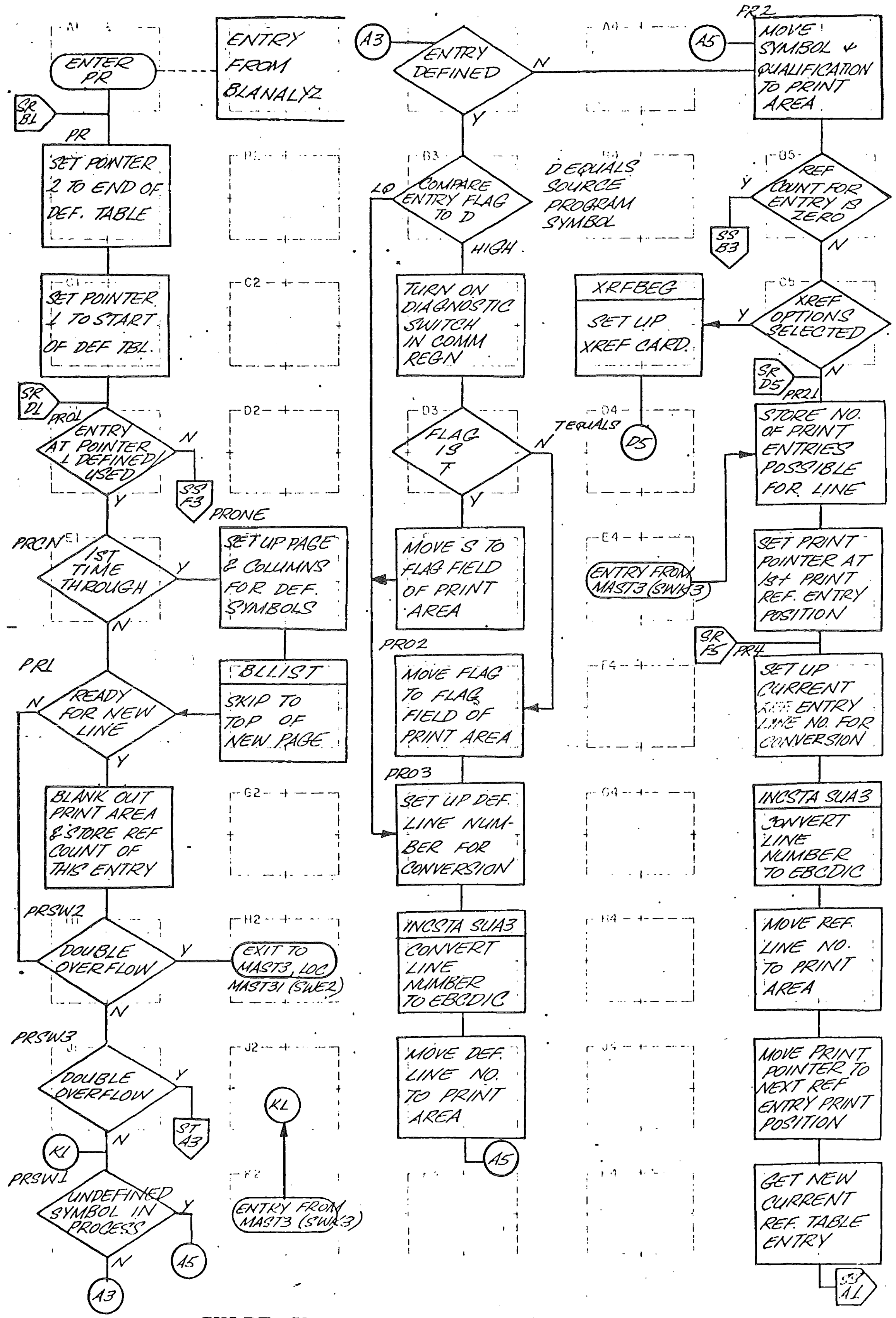

ENTER PR
ENTRY FROM BLANALYZ
SR BL
PR
SET POINTER 2 TO END OF DEF. TABLE
SET POINTER 1 TO START OF DEF TBL.
SR D1
PR01
ENTRY AT POINTER 1 DEFINED/USED
SS F3
PRONE
PRON
1ST TIME THROUGH
SET UP PAGE & COLUMNS FOR DEF. SYMBOLS
PR1
READY FOR NEW LINE
BLLIST
SKIP TO TOP OF NEW PAGE
BLANK OUT PRINT AREA & STORE REF COUNT OF THIS ENTRY
PRSW2
DOUBLE OVERFLOW
EXIT TO MAST3, LOC MAST31 (SWE2)
PRSW3
DOUBLE OVERFLOW
ST A3
K1
PRSW1
UNDEFINED SYMBOL IN PROCESS
A5
A3
ENTRY FROM MAST3 (SWE3)
ENTRY DEFINED
COMPARE ENTRY FLAG TO D
LO
HIGH
D EQUALS SOURCE PROGRAM SYMBOL
TURN ON DIAGNOSTIC SWITCH IN COMM REGN
FLAG IS T
T EQUALS
MOVE S TO FLAG FIELD OF PRINT AREA
PR02
MOVE FLAG TO FLAG FIELD OF PRINT AREA
PR03
SET UP DEF. LINE NUMBER FOR CONVERSION
INCSTA SUA3
CONVERT LINE NUMBER TO EBCDIC
MOVE DEF. LINE NO. TO PRINT AREA
XRFBEG
SET UP XREF CARD
D5
ENTRY FROM MAST3 (SWE3)
PR2
MOVE SYMBOL + QUALIFICATION TO PRINT AREA
REF COUNT FOR ENTRY IS ZERO
SS B3
XREF OPTIONS SELECTED
SR D5
PR21
STORE NO. OF PRINT ENTRIES POSSIBLE FOR LINE
SET PRINT POINTER AT 1st PRINT REF. ENTRY POSITION
SR F5
PR4
SET UP CURRENT REF ENTRY LINE NO. FOR CONVERSION
INCSTA SUA3
CONVERT LINE NUMBER TO EBCDIC
MOVE REF LINE NO. TO PRINT AREA
MOVE PRINT POINTER TO NEXT REF ENTRY PRINT POSITION
GET NEW CURRENT REF. TABLE ENTRY
SS A1

CHART SR. PR ROUTINE (SHEET 1 OF 3)

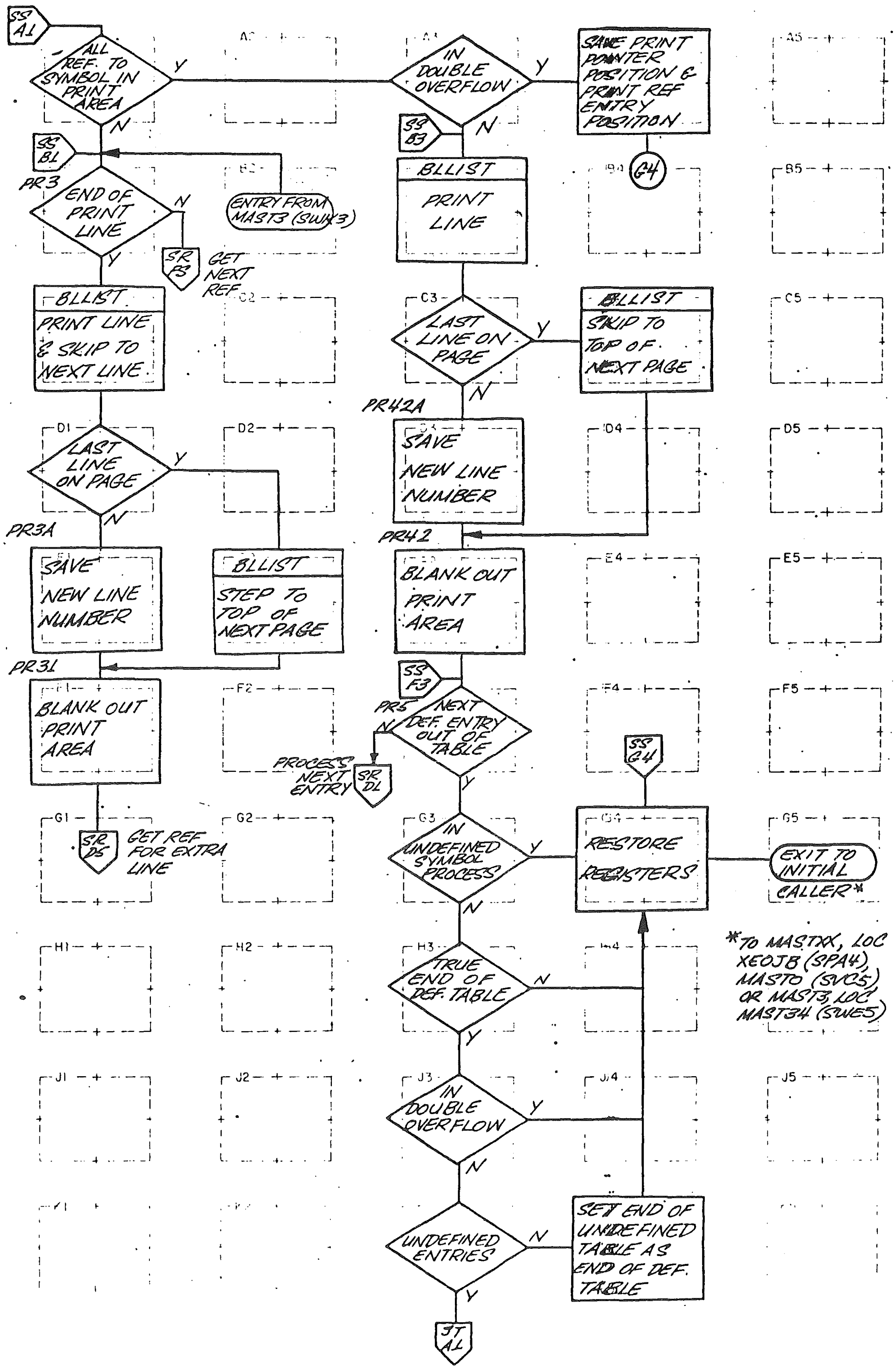


CHART SS. PR ROUTINE (SHEET 2 OF 3)

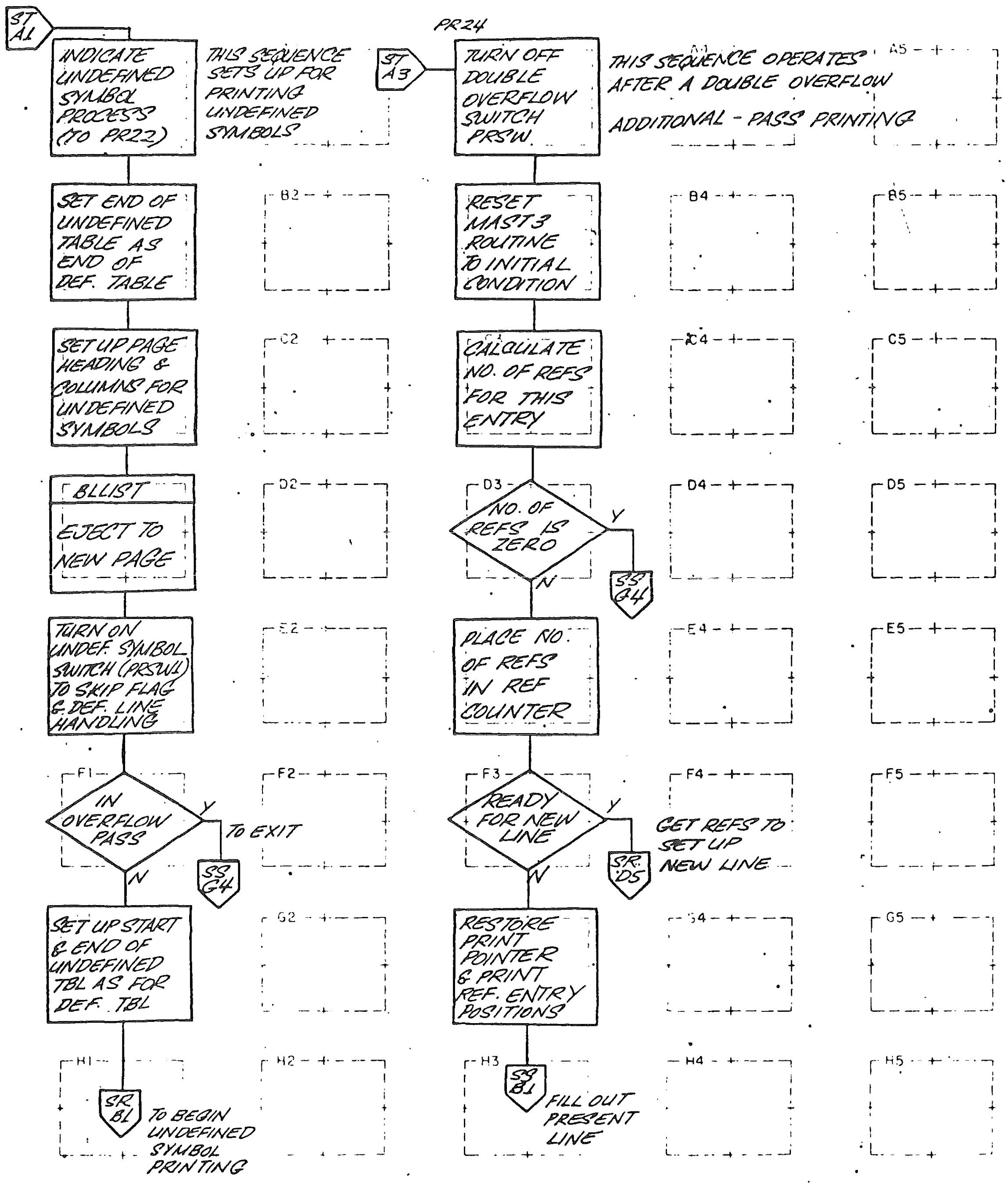

ST A1
INDICATE UNDEFINED SYMBOL PROCESS (TO PR22)
THIS SEQUENCE SETS UP FOR PRINTING UNDEFINED SYMBOLS
SET END OF UNDEFINED TABLE AS END OF DEF. TABLE
SET UP PAGE HEADING & COLUMNS FOR UNDEFINED SYMBOLS
BLLIST
EJECT TO NEW PAGE
TURN ON UNDEF. SYMBOL SWITCH (PRSWU) TO SKIP FLAG & DEF. LINE HANDLING
IN OVERFLOW PASS
Y
TO EXIT
SS G4
N
SET UP START & END OF UNDEFINED TBL AS FOR DEF. TBL
SR B1
TO BEGIN UNDEFINED SYMBOL PRINTING
PR24
ST A3
TURN OFF DOUBLE OVERFLOW SWITCH PRSW
THIS SEQUENCE OPERATES AFTER A DOUBLE OVERFLOW ADDITIONAL - PASS PRINTING
RESET MAST3 ROUTINE TO INITIAL CONDITION
CALCULATE NO. OF REFS FOR THIS ENTRY
NO. OF REFS IS ZERO
Y
SS G4
N
PLACE NO. OF REFS IN REF COUNTER
READY FOR NEW LINE
Y
SR D5
GET REFS TO SET UP NEW LINE
N
RESTORE PRINT POINTER & PRINT REF. ENTRY POSITIONS
SS B1
FILL OUT PRESENT LINE

CHART ST. PR ROUTINE (SHEET 3 OF 3)

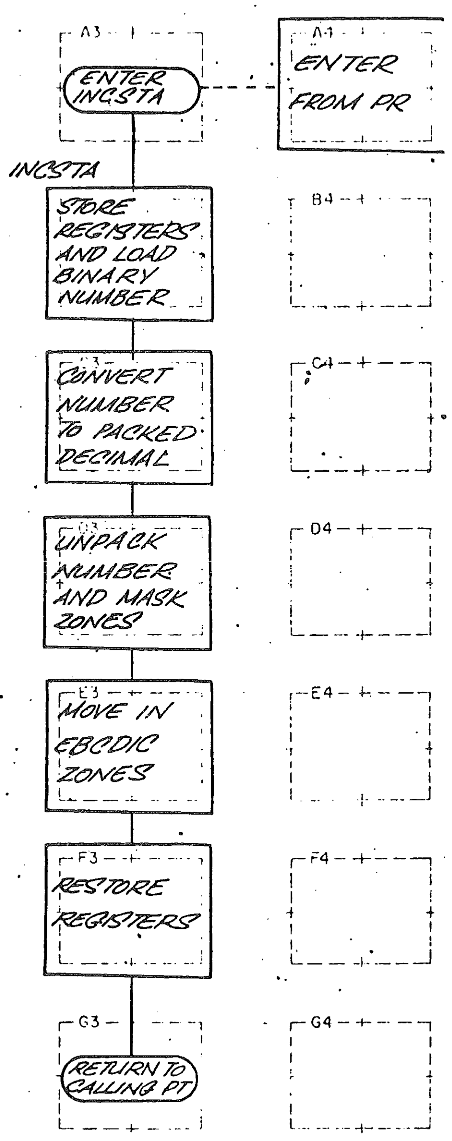
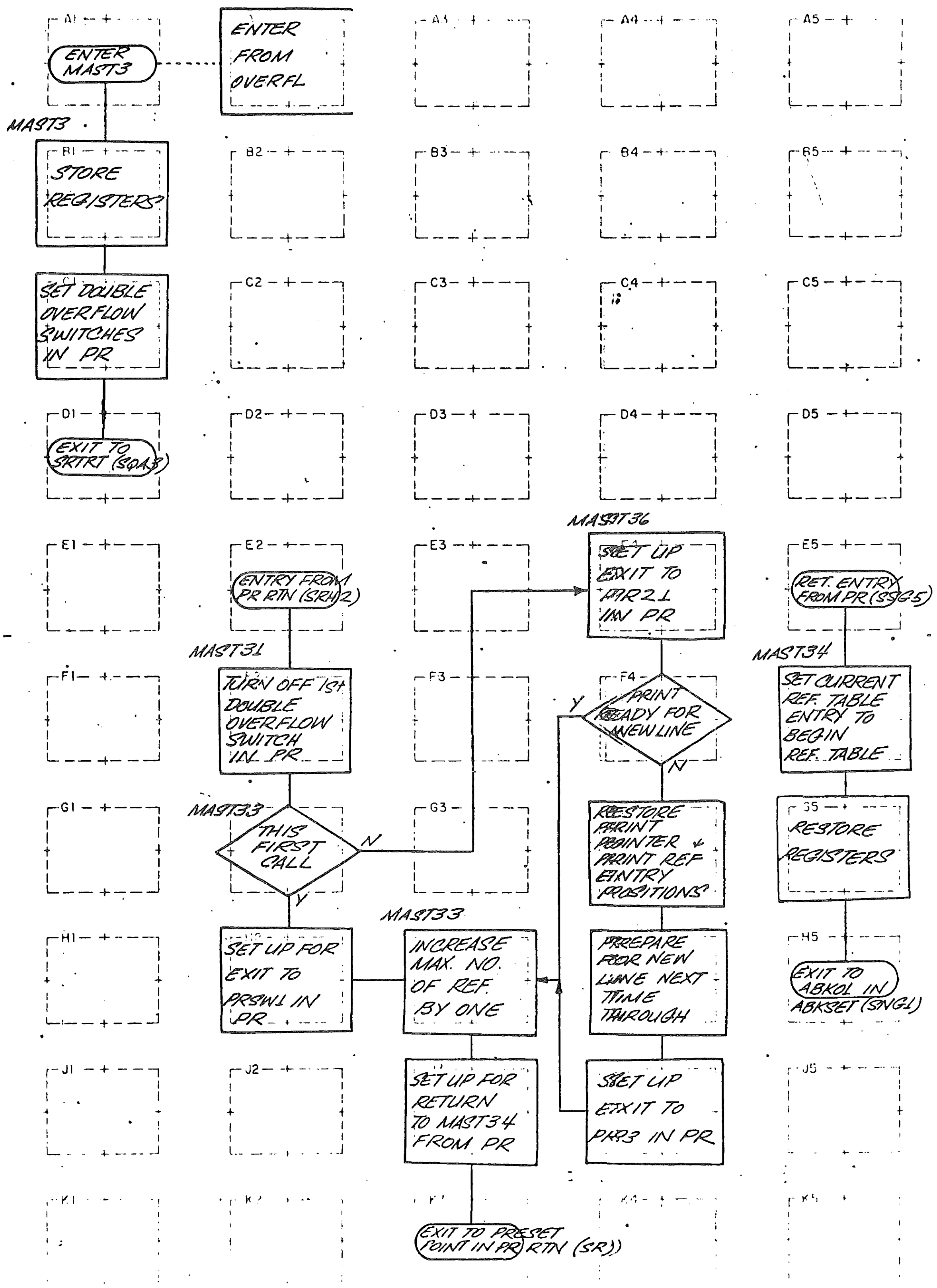
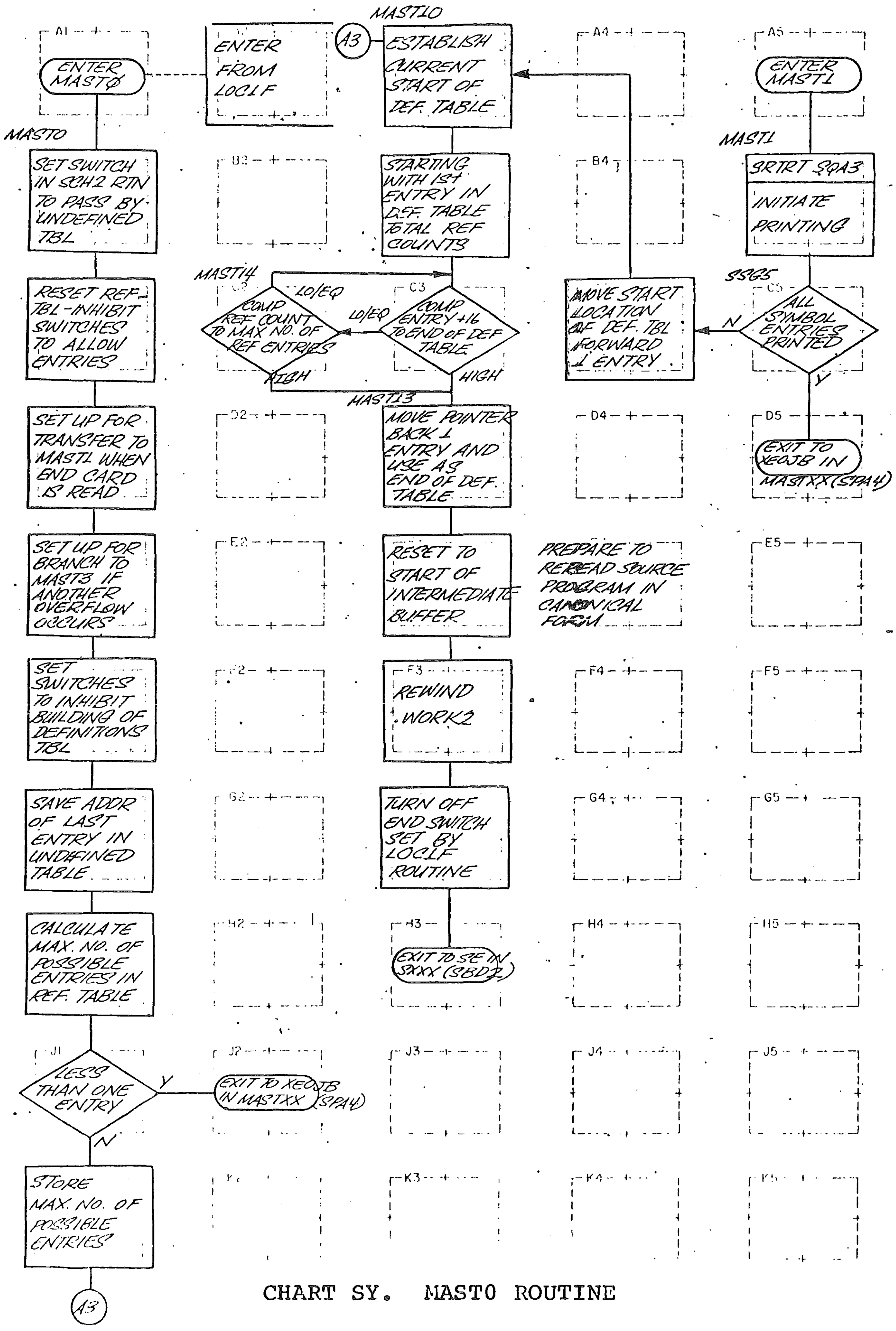


CHART SU. INCSTA SUBROUTINE



ENTER MAST3
ENTER FROM OVERFL
MAST3
STORE REGISTERS
SET DOUBLE OVERFLOW SWITCHES IN PR
EXIT TO SRTRT (SQA5)
ENTRY FROM PR RTN (SRH2)
MAST31
TURN OFF 1st DOUBLE OVERFLOW SWITCH IN PR
MAST33
THIS FIRST CALL
N
Y
SET UP FOR EXIT TO PRSW1 IN PR
MAST33
INCREASE MAX. NO. OF REF BY ONE
SET UP FOR RETURN TO MAST34 FROM PR
EXIT TO PRESET POINT IN PR RTN (SRJ)
MAST36
SET UP EXIT TO PR21 IN PR
PRINT READY FOR NEW LINE
Y
N
RESTORE PRINT POINTER + PRINT REF ENTRY POSITIONS
PREPARE FOR NEW LINE NEXT TIME THROUGH
SET UP EXIT TO PR3 IN PR
RET. ENTRY FROM PR (SSG5)
MAST34
SET CURRENT REF. TABLE ENTRY TO BEGIN REF. TABLE
RESTORE REGISTERS
EXIT TO ABKOL IN ABKSET (SNG1)

CHART SW. MAST3 ROUTINE



ENTER MASTØ
ENTER FROM LOCLF
MASTO
SET SWITCH IN SCH2 RTN TO PASS BY UNDEFINED TBL
RESET REF-TBL-INHIBIT SWITCHES TO ALLOW ENTRIES
SET UP FOR TRANSFER TO MAST1 WHEN END CARD IS READ
SET UP FOR BRANCH TO MAST3 IF ANOTHER OVERFLOW OCCURS
SET SWITCHES TO INHIBIT BUILDING OF DEFINITIONS TBL
SAVE ADDR OF LAST ENTRY IN UNDEFINED TABLE
CALCULATE MAX. NO. OF POSSIBLE ENTRIES IN REF. TABLE
LESS THAN ONE ENTRY
Y
N
EXIT TO XEQJB IN MASTXX (SPA4)
STORE MAX. NO. OF POSSIBLE ENTRIES
A3
MAST10
A3
ESTABLISH CURRENT START OF DEF. TABLE
STARTING WITH 1ST ENTRY IN DEF. TABLE TOTAL REF COUNTS
MAST14
LO/EQ
COMP REF COUNT TO MAX NO. OF REF ENTRIES
HIGH
LO/EQ
COMP ENTRY+16 TO END OF DEF TABLE
HIGH
MAST13
MOVE POINTER BACK 1 ENTRY AND USE AS END OF DEF. TABLE
RESET TO START OF INTERMEDIATE BUFFER
PREPARE TO REREAD SOURCE PROGRAM IN CANONICAL FORM
REWIND WORK2
TURN OFF END SWITCH SET BY LOCLF ROUTINE
EXIT TO SE IN SXXX (SBD2)
MOVE START LOCATION OF DEF. TBL FORWARD 1 ENTRY
N
ENTER MAST1
MAST1
SRTRT SQA3
INITIATE PRINTING
SSG5
ALL SYMBOL ENTRIES PRINTED
Y
EXIT TO XEQJB IN MASTXX (SPA4)

CHART SY. MASTO ROUTINE

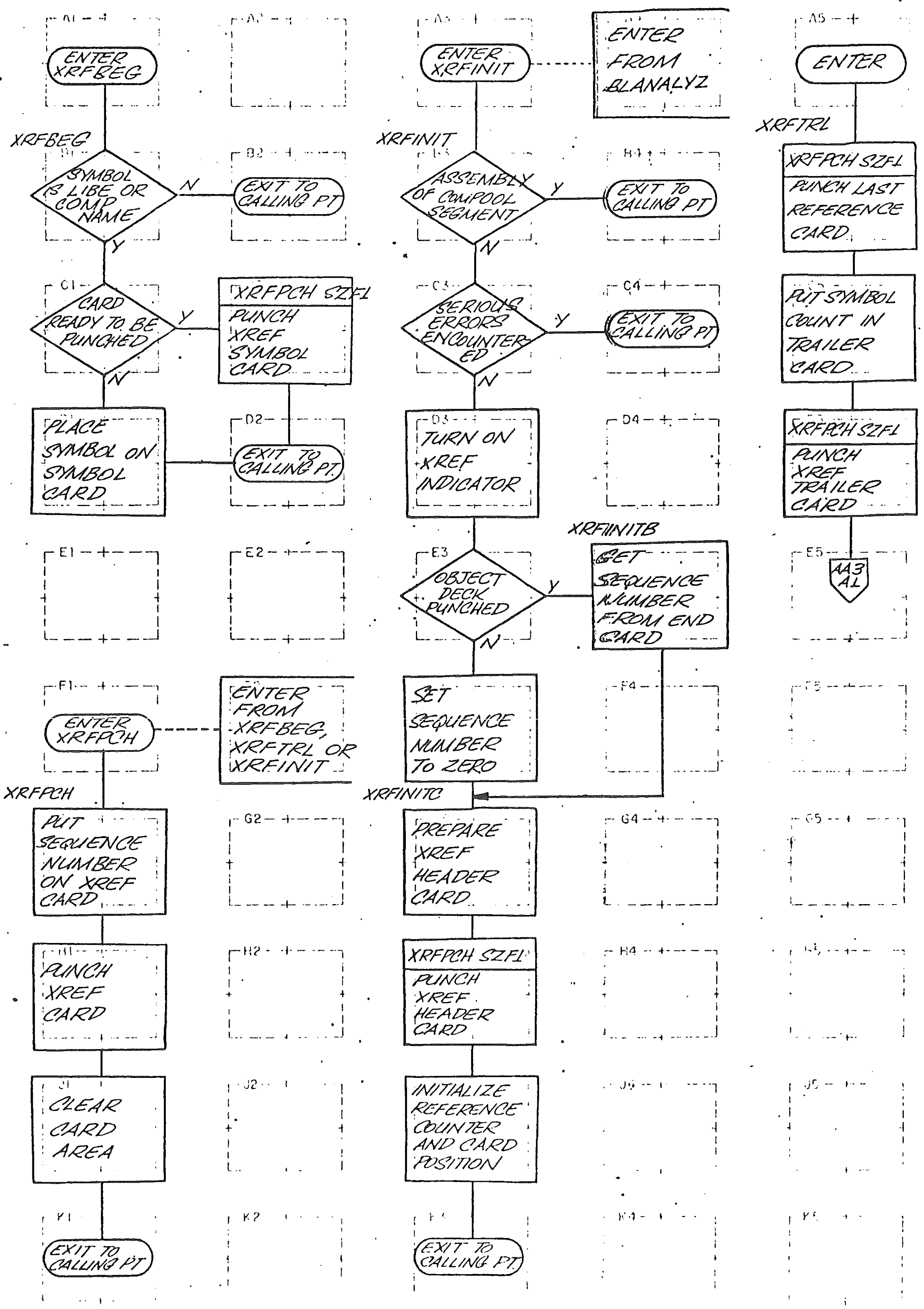


CHART SZ. XREF ROUTINE

5.0 STORAGE AND TIMING

5.1 STORAGE REQUIREMENTS

The BAL assumbler can operate in a minimum environment of one Storage Element (SE) (32K words). Although it can be run in any larger configuration, it will use a maximum of eight SEs. Any additional storage beyond 8 SEs will simply be ignored.

The actual storage required to assemble a given program is dependent on the availability of WORK2 and on the size of the program. Accompanying tables 5-1 and 5-2 give the maximum program parameters that can be accommodated in each memory configuration with and without WORK2, respectively.

Figure 5-1 illustrates the storage layout of the assemblies that comprise the BAL assembler. Figure 5-2 summarizes the layout of the buffers and variable length tables that are used by BAL.

5.2 TIMING

There is no simple formula to determine the timing for the BAL assembler, as it is highly dependent on machine configuration, especially storage and WORK2 availability. However, for any given configuration, the time for a non-compool assembly will be roughly proportional to the number of input statements.

When a compool is involved, timing will be proportional to number of statements from SYSIN plus number of statements actually processed from the compool, plus an overhead factor representing tape time to find the first compool segment and tape time to skip unwanted segments.

Use of the PUNCHC option will add time for an additional pass over the compool tape, plus assembly time for those statements selected from the compool.

TABLE 5-1. ASSEMBLER SES VS VARIABLES (.WORK2 AVAILABLE)

Variables	Number of SEs							
	1	2	3	4	5	6	7	8
Maximum number of symbols	1024	4096	8192	8192	16384	16384	16384	20480
Maximum number of address constants	512	2048	4096	4096	8192	8192	8192	8192
Maximum number of literals.	128	1024	2048	2048	4096	4096	4096	4096

TABLE 5-2. ASSEMBLER SE VS VARIABLES (.WORK2 NOT AVAILABLE)

Variables	Number of SEs							
	1	2	3	4	5	6	7	8
Maximum number of symbols	144	719	1296	1872	2248	3023	3600	4176
Maximum number of address constants	72	359	646	934	1221	1509	1796	2048
Maximum number of literals	72	359	646	934	1221	1509	1796	2048
Approximate number cards	478	2778	5078	7378	9678	11978	14278	16578

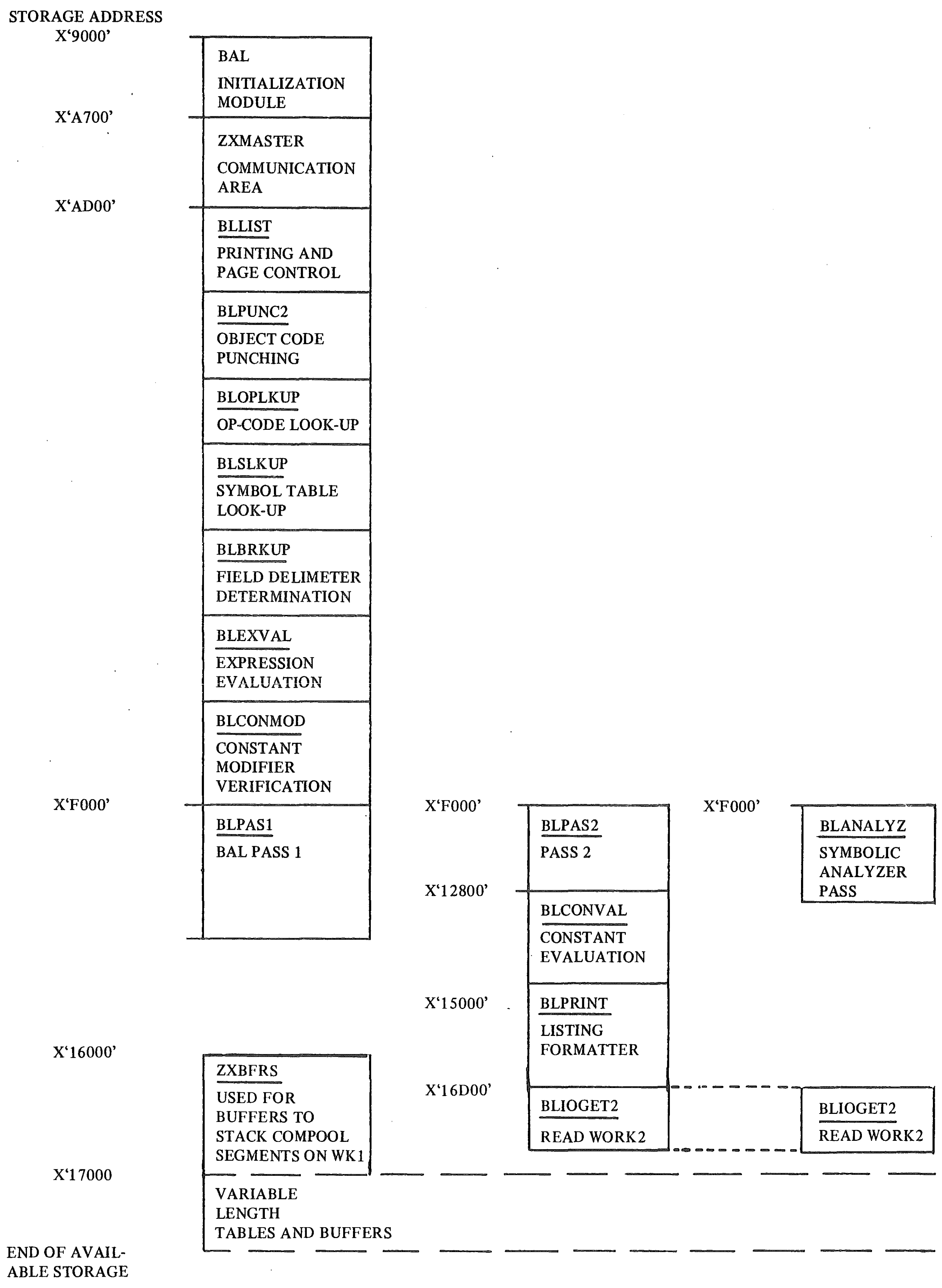


FIGURE 5-1. BAL PROGRAM STORAGE REQUIREMENTS

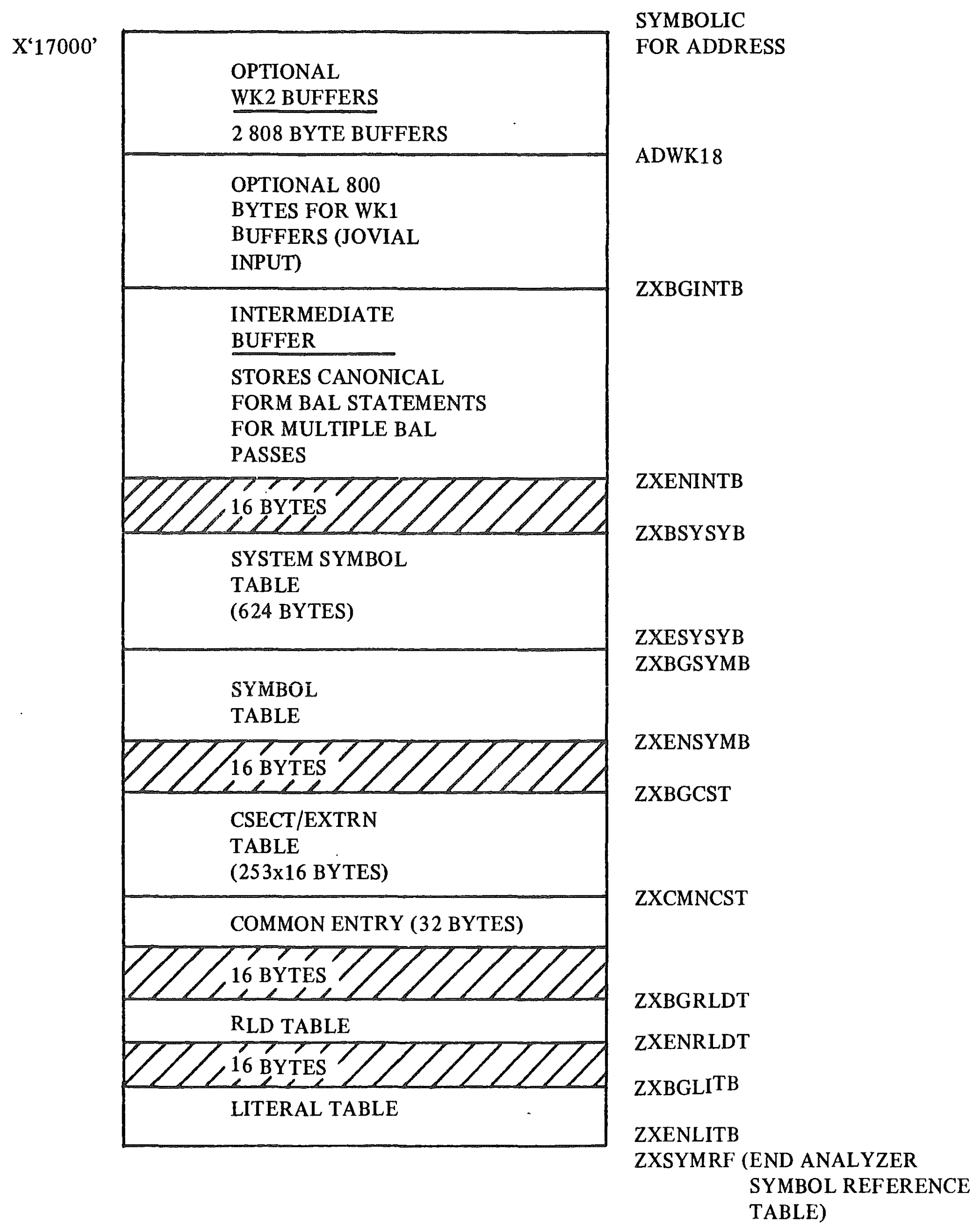


FIGURE 5-2. BAL BUFFERS AND TABLES

6.0 DATA SPECIFICATIONS

6.1 TABLE FORMATS

6.1.1 Symbol Table

The symbol table is built by BLPAS1 and used by BLPAS2 and BLANALYZ. Every symbol defined in the source program creates an entry in this table in the format shown in Figure 6-1.

6.1.2 System Symbol Table

The system symbol table, containing permanently defined system symbols, is appended to the end of the symbol table when the symbol table is set up by the INIT1 routine of BAL. Entries in the system symbol table have the same format as those in the symbol table (see Figure 6-1).

6.1.3 CSECT-EXTRN Table

The CSECT-EXTRN table is built by BLPAS1 and updated during its interlude. Entries are made for CSECT, DSECT, COM, and EXTRN statements in the format given in Figure 6-2.

6.1.4 ENTRY Table

The ENTRY table is built by BLPAS1 from symbols designated by ENTRY statements. All defined ENTRY symbols (i.e., those also entered in the symbol table) are put out during the interlude as ESD cards. Each entry in the ENTRY table contains a symbol in an eight-byte field and its qualifier in a one-byte field. The format of an entry is shown in Figure 6-3.

6.1.5 Literal Table

The literal table is built by BLPAS1 and emptied after an LTORG statement and/or after the END statement. Each literal being put out as DCL statement for BLPAS2 is assigned a location by entering the literal dummy name in the symbol table. The format of an entry in the literal table is shown in Figure 6-4.

6.1.6 RLD Table

The RLD table is built by BLPAS2 to develop the relocation dictionary for the object program. Each entry in the table is in the format shown in Figure 6-5.

Field	Symbol	Qual	Location	Reloc ID	Length	Control	Type	Scale
Byte No.	0 7	8	9 11	12	13	14		15
Bytes	8	1	3	1	1	1/2	1/2	1

where:

Symbol - EBCDIC characters, left-justified

Qual - EBCDIC character

Location - address assigned to symbol

Reloc ID - 0 = absolute; 1 through 254 = CSECT or DSECT ID; 255 = Common ID

Length - one less than length defined or implied, in bytes

Control - 0 = empty; 1 = defined; 3 = multi-defined; 5 = symbol is TEQUATED, re-evaluate the equate

Type -

0 =	X (hexadecimal)
1 =	C (EBCDIC)
2 =	P (packed decimal)
3 =	Z (zoned decimal)
4 =	H (fixed point halfword)
5 =	F (fixed point word)
6 =	E (floating point, singleword)
7 =	D (floating point, doubleword)
8 =	I (instruction label)
9 =	A (address constant, word)
10 =	S (base and displacement, halfword)

Scale Factor - Binary (2's complement) 10100000 to 10011111 (-96 to +159 decimal)

FIGURE 6-1. SYMBOL TABLE ENTRY FORMAT

Field	Symbol	Current Location	Highest Location	Reloc ID	ESD Type
Byte No.	0-7	8-10	11-13	14	15
Bytes	8	3	3	1	1

where:

Symbol - Section name or EXTRN symbol, EBCDIC characters

Current Location - Location counter for section, 0 through $2^{24}-1$

Highest Location - Location counter for current end of section, 0 through $2^{24}-1$

Reloc ID - see definition under symbol table (Figure 6-1)

ESD Type - C = CSECT; D = DSECT; X'02' = EXTRN; X'0A' = COM

FIGURE 6-2. CSECT-EXTRN TABLE ENTRY FORMAT

Field	Symbol	Qual
Byte No.	0	8
Bytes	8	1

FIGURE 6-3. ENTRY TABLE ENTRY FORMAT

Field	Source Length	Dummy Name	Object Length	Source Format
Byte No.	0	1-8	9	10-n
Bytes	1	8	1	variable

where:

Source Length - length in bytes of literal source format

Dummy Name - =000 followed by a four-byte identifying number, assigned in order of entry (= is X'7E')

Object Length - one less than the literal object length, in bytes

Source Format - the literal as written in source program

FIGURE 6-4. LITERAL TABLE ENTRY FORMAT

Field	Location	Reloc ID 1	Reloc ID 2	Sign	Length
Byte No.	0-3	4	5	6	7
Bytes	4	1	1	1	1

where:

Location - storage location of address constant

Reloc ID1 - ID of address constant (ID of section containing the Adcon)

Reloc ID2 - ID of symbol in address constant (ID of section containing the symbol)

Sign - addition or subtraction of symbolic in address constant

Length - length of address constant (1, 2, 3, or 4 bytes)

FIGURE 6-5. RLD TABLE ENTRY FORMAT

6.1.7 USING Table

The USING table records for BLPAS2 the general registers that are available as base registers in any given segment of a source program and the value that is held in each base register. The table contains 17 entries, two for register 0 and one each for every other general register. The first entry for register 0 is always available with a value of zero and is unaffected by USING or DROP statements. All remaining entries are affected by USING and DROP statements, except that the switchable entry for register 0 always has a value of zero. The format of each entry in the USING table is shown in Figure 6-6.

Field	Avail	Reloc ID	Reg No.	Reg No.	Value
Byte No.	0	1	2	3	4-7
Bytes	1	1	1	1	4

where:

Avail - 1 = available for use as base register; 0 = not available

Reloc ID - ID of symbol in expression for base value

Reg No. - general register number in EBCDIC

Reg No. - general register number in hexadecimal

Value - base value for computation of displacement

FIGURE 6-6. USING TABLE ENTRY FORMAT

6.1.8 Operation Code Table

The operation code table, contained in BLOPLKUP, contains an entry for every machine operation code and every valid pseudo-operation code (as well as for some invalid pseudo-operation codes). These entries are arranged alphabetically by mnemonic, but the entries are of two types: one for machine operation codes and another for pseudo-operation codes. The table is used by both BLPAS1 and BLPAS2. The format for a machine operation code entry is shown in Figure 6-7.

Field	Mnemonic	Obj	Type 1	Align 1	Type 2	Align 2	Type 3	Align 3	Oper Count	Operation Type
Byte No.	0-4	5	6	7	8	9	10	11	12-14	15
Bytes	5	1	1	1	1	1	1	1	3	1

where:

Mnemonic - operation code mnemonic, left-justified, in EBCDIC

Obj - object code in hexadecimal

Type - operand format type (number designates operand): 0 = no operand; 4 = register; 8 = storage address; 12 = indexable storage address; 16 = storage address with 4-bit length field; 20 = storage address with 8-bit length field; 24 = 8-bit integer; 28 = storage address (SI) with I_2 field zeroed; 32 = storage address with zeroed 4-bit length field; 36 = shift count; 40 = R1 register operand with R2 zero.

Align - alignment code for operand (number designates operand): code 0 = no alignment; 1 = halfword; 2 = word; 3 = doubleword. If type is a register operand: 0 = no special register; 1 = only even register; 4 = floating point register.

Oper Count - number of operands required by the instruction.

Operation Type - non-privileged (code = 0) or privileged (code = 1)

FIGURE 6-7. MACHINE OPERATION ENTRY FORMAT

The format for a pseudo-operation code entry is shown in Figure 6-8.

Field	Mnemonic	Obj	N/A	Mask	BLPAS1 Pointer	BLPAS2 Pointer
Byte No.	0-4	5	6	7	8-11	12-15
Bytes	5	1	1	1	4	4

where:

Mnemonic - operation code mnemonic, left-justified, in EBCDIC

Obj - hex 00, identifying entry as pseudo-operation

N/A - not used

Mask - used for extended branch mnemonics

BLPAS1 Pointer - pointer to processing subroutine in BLPAS1 for this pseudo-operation

BLPAS2 Pointer - pointer to processing subroutine in BLPAS2 for this pseudo-operation

FIGURE 6-8. PSEUDO-OPERATION ENTRY FORMAT

6.1.9 Definition Table

The definition table is built in alphabetical order by the analyzer from symbols in the symbol and system symbol tables. During analyzer processing, line number, flags, and count (number of references to symbol), are added. The format of this table is shown in Figure 6-9.

6.1.10 Reference Tables

The reference table is built during analyzer processing. The format of entries made in this table is shown in Figure 6-10.

6.1.11 Undefined Table

The undefined table is built during analyzer processing. The format of entries made in this table is shown in Figure 6-11.

Field	Symbol	Qual	Flag	Line No.	Count
Byte No.	0-7	8	9	10-12	13-15
Bytes	8	1	1	3	3

where:

Symbol - EBCDIC characters, left-justified with trailing blanks

Qual - EBCDIC character (blank if no qualification specified for symbol)

Flag -	
	A = an undefined symbol (not printed)
	D = a defined symbol (not printed)
	M = a multi-defined symbol
	R = a multi-redefined system symbol
	S = any system symbol (an S flag, as such, is not printed out)
	T = a redefined system symbol (printed out as S)

Line No. - line number of the assembly listing statement in which the symbol is defined

Count - number of times the symbol is used in operands

FIGURE 6-9. DEFINITION TABLE ENTRY FORMAT

Field	Reference	Line No.
No. of Bytes	4	4

where:

Reference - address of the symbol's entry in the Definition Table - or, if a symbol is undefined, an undefined code that indicates an entry in the Undefined Table.

Line No. - number of line in the assembly listing in which the symbol appears in the operand field

FIGURE 6-10. REFERENCE TABLE ENTRY FORMAT

Field	Symbol	Qual	N/A	Undefined Code	Count
Byte No.	0-7	8	9	10-12	13-15
Bytes	8	1	1	3	3

where:

Symbol - EBCDIC characters, left-justified with trailing blanks

Qual - EBCDIC character

N/A - not used

Code - the undefined code is a sequential number, beginning with one, assigned to undefined symbols as they are discovered

Count - number of times the symbol is used in operands

FIGURE 6-11. UNDEFINED TABLE ENTRY FORMAT

6.1.12 Segment Table

The segment table is built and used by BLPAS1. If the PUNCHC option is specified, it is written out on WORK1 at the end of BLPAS1 and read in after completion of BLPAS2 and BLANALYZ to allow copying of the compool to WORK1.

Each entry in the table corresponds to a PSEG segnam statement. The format of entries made in this table is shown in Figure 6-12.

Field	Flag	Length	Name
Byte No.	0	1	2-7
Bytes	1	1	6

where:

Flag - 0 = entry not used; 1 = segment waiting to be read; 2 = segment already read or in error; X'FF' = end of table.

Length - length minus one (in characters) of segment name (suitable for EX'ing CLC, etc.)

Name - segment name, left-justified with trailing blanks.

FIGURE 6-12. SEGMENT TABLE ENTRY FORMAT

6.2 INTERNAL DATA FORMATS

These records are built by BLPAS1 and placed in the intermediate work file. They are read and processed by both BLPAS2 and BLANALYZ.

6.2.1 Statement Record

Statements in canonical form are in the format shown in Figure 6-13.

6.2.2 Ignore Statement Record

Ignore statement records are identical in format to statement records except that their first character is #(X'7B') and their length is accordingly one byte longer. The format of a statement record is shown in Figure 6-13.

Ignore statement records are issued during BLPAS1 for certain erroneous statements. BLPAS2 will not process these records but they will be printed on the listing. The error is indicated by a diagnostic record which precedes the ignore statement record.

Comment records are in the format shown in Figure 6-14.

6.2.3 Diagnostic Record

The need for diagnostics necessitates a special record that is put out during BLPAS1. BLPAS2 presents these records, as well as its own diagnostics, to a routine that will print the diagnostic. The format of a diagnostic record is shown in Figure 6-15.

6.2.4 Literal Reference Records

Literal reference records are issued in BLPAS1 for every statement containing a literal. The format of literal reference records is shown in Figure 6-16.

Field	L_1	L_2	L_3	L_4	Symbol	Operation	Operand-Comments	Cols. 72-80
Byte No.	0	1	2	3	*	*	*	*
Bytes	1	1	1	1	0-8	1-5	0-69	9

where:

L_1 - is the byte length of the entire record in binary ($14 \leqslant L_1 \leqslant 85$)

L_2 - is the byte length of the symbol ($0 \leqslant L_2 \leqslant n$) where n is the number of non-blank characters in the field

L_3 - is the byte length of the operation code ($0 \leqslant L_3 \leqslant n$)

L_4 - is the byte length of the operand-comment field with all but one trailing blank suppressed ($0 \leqslant L_4 \leqslant 69$)

Note: In the fields following L_4, * indicates a field of variable length and position.

Symbol - EBCDIC characters, left-justified with blanks suppressed

Operation Code - EBCDIC characters, left-justified with blanks suppressed

Operand-Comments - EBCDIC characters with all but two trailing blanks suppressed. If a quote (') appears in the operand, the entire operand field is included

Cols. 72-80 - appear exactly as punched on the card. A non-blank in column 72 is ignored

FIGURE 6-13. STATEMENT RECORD FORMAT

Field	*	Columns 2-71 as on original statement	Columns 72-80
Byte No.	0	1-70	71-79
Bytes	1	70	9

Note that the first character of this record is *(X'5C').

FIGURE 6-14. COMMENT RECORD FORMAT

Field	'	Code	Symbol
Byte No.	0	1-4	5-12
Bytes	1	4	8

Note that the first character of the record is '(X'7D').

where:

Code - a four-byte code identified with the diagnostic message to be printed (see Subsection 6.4)

Symbol - the symbol will be printed along with the diagnostic message, if a symbol is involved

FIGURE 6-15. DIAGNOSTIC RECORD FORMAT

Field	=	Dummy Name	Source Length
Byte No.	0	1-8	9
Bytes	1	8	1

Note that the first character of the record is = (X'7E').

where:

Dummy Name - an 8-character, left-justified name assigned to the literal appearing in the next statement

Source Length - a byte indicating the number of characters that are used on the source card for the literal

FIGURE 6-16. LITERAL REFERENCE RECORD FORMAT

6.3 ROUTINE INPUT/OUTPUT FORMATS

Length, SYM Type, Scale Factor, and Control are described in the symbol table and are taken from the first element (symbol or integer) encountered in the expression.

Location is the value of the expression.

Reloc ID Sign - a one if minus is associated with relocatable symbol.

Reloc ID is either zero, if absolute expression, or nonzero ID if relocatable.

Error type - this is set by bits in the following manner.

a. Bit 31 is set if expression is complexly relocatable

b. Bit 30 is set if expression may be in error

c. Bit 29 is set if rest of statement cannot be evaluated

d. Bit 28 is set if expression is partially evaluated

e. Bit 27 is set if expression is multi-defined

Error Count - this is a count of the number of diagnostics issued for the expression.

Diagnostic - following the count are the diagnostics issued (for description see ''Diagnostic Record'').

6.3.1 BLCONMOD Routine Format

a. Input

   Input Loc - The address of the first character of the data component

   End Loc - The address of the last character of the operand field

b. Output

   On return from BLCONMOD, register 1 will contain the address of a table which contains the following.

   1. Address of 1st character in the value list - one word. Normally will reference a (, quote, or blank

   2. Address of the last character of the value list - one word. Normally will reference a), quote, or blank

   3. Multiplicity - one word

   4. Length of first element of constant in bits - binary word

   5. Total length of constant in bytes - binary word

   6. Type - one byte

      Length of first element in value list in bytes (-1) - one byte (value suitable for use in EXecuting CLC, MVC, etc.)

Alignment code - one byte
1 = byte
2 = halfword
4 = word
8 = doubleword

Scale factor - one byte

7. Error count - one word

8. Diagnostic code 1 - one word (right-justified)

9-10. Error symbol - two words

11. Diagnostic code 2 - one word (right-justified)

12-13. Error symbol - two words

6.3.2 BLCONVAL Routine Format

a. Input

The starting address of the multiplicity, type, length and scale factor.

b. Output

This routine uses the set of values (multiplicity, type, length, etc.) determined in BLCONMOD to evaluate the constant list. The routine requires that a general register contain the starting address of the above values.

On return from BLCONVAL, another general register will contain the address of a table containing the following.

1. Address of the assembled constant - one word

2. Length of constant (after padding, excluding multiplicity) - one word

3. Error count - one word

4. Diagnostic code - one word (right-justified)

5. Error symbol - two words

6.3.3 FROMD Routine Format

a. Input (Figure 6-17)

The symbolic name of this table is TABLEA.

b. Output (Figure 6-18)

The symbolic name of this table is TABLEA

6.3.4 FROMF Routine Format

a. Input (Figure 6-19)

The symbolic name of this is TABLEA.

b. Output (Figure 6-20)

The symbolic name of this is TABLEA

6.3.5 BLEXVAL Routine

a. Input

Register 1 contains the beginning address of the expression.

b. Output

On return from BLEXVAL, register 1 has the address of an output table and register 2 has the address of the expression terminator. The output table contains the following.

Word	Contents	Position
Word 0	Location (value)	
Word 1	Length	
Word 2	SYM Type	
Word 3	Scale Factor	
Word 4	Control	right-justified
Word 5	Reloc ID Sign	1st halfword (0=plus, 1=minus)
	Reloc ID	2nd halfword (right-justified)
Word 6	Error Type	right-justified
Word 7	Error Count	
Word 8	Diagnostic Code	right-justified
Word 9-10	Error Symbol	left-justified
Word 11	Diagnostic Code 2	
Word 12-13	Error Symbol 2 etc.	

Contents	SF	E	N	Number
Byte No.	0	4	8	12-35
Bytes	4	4	4	24

where:

SF - Scale factor, if specified, $0 \leq SF < +14$. Each unit of the scale factor shifts the fraction part of the converted number 4 bits to the right and increases the exponent by one.

E - Exponent (signed integer)

N - Number of characters to the left of the decimal point. Algebraic sum of E and N must be $\leq +76$ and ≥ -78.

Number - EBCDIC representation of the number to be converted

FIGURE 6-17. FROMD INPUT TABLE ENTRY FORMAT

Contents	F	Representation
Byte No.	0	4
Bytes	4	8

where:

F - 0 = successful conversion
-1 = overflow
-2 = underflow

Representation - double-precision floating-point representation of the value constant.

FIGURE 6-18. FROMD OUTPUT TABLE ENTRY FORMAT

Contents	L_1	L_2	SF	E	N	S	Number
Byte No.	0	4	8	12	16	20	21
Bytes	4	4	4	4	4	1	23

where:

L_1 - length in bytes of the input number

L_2 - length in bits of input number

SF - scale factor

E - exponent

N - number of digits to left of decimal point

S - sign of the input number

Number - input number in EBCDIC

FIGURE 6-19. FROMF INPUT TABLE ENTRY FORMAT

Contents	F	L	Binary Constant
Byte No.	0	2	4
Bytes	2	2	8

where:

F - indicates an abnormal condition

LE	-	invalid length or scale
IN	-	lost integer
FR	-	lost fraction
LT	-	low-order truncation of an integer
ƀƀ	-	absence of flag indicating normal output

L - length in bits of binary constant

Binary Constant - binary representation of decimal number specified by BLCONVAL

FIGURE 6-20. FROMF OUTPUT TABLE ENTRY FORMAT

6.4 DIAGNOSTIC MESSAGES

Code	Messages
1	FIELD n HAS INVALID PUNCTUATION
2	FIELD n HAS INVALID CHARACTER FOUND
3	FIELD n HAS A SYMBOL OR NUMBER WHICH IS TOO LONG
4	FIELD n HAS AN EXPRESSION WHICH IS LONG OR COMPLEX
5	(symbol) IS AN UNDEFINED SYMBOL
6	FIELD n HAS AN INVALID USE OF *
7	FIELD n IS INVALIDLY COMPLEX RELOCATABLE
8	FIELD n HAS A VOID EXPRESSION - POSSIBLE ERROR
9	FIELD n HAS BEEN TRUNCATED - POSSIBLE ERROR
10	FIELD n HAS A RELOCATABLE SYMBOL WHICH IS MULTIPLIED OR DIVIDED
11	FIELD n HAS TOO MANY ELEMENTS IN AN EXPRESSION
12	(symbol) IS A MULTI-DEFINED SYMBOL
13	USE OF PRIVILEGED OPERATION CODE - POSSIBLE ERROR
14	FIELD n HAS AN EXPRESSION INVALIDLY TERMINATED
15	PSEUDO-OPERATION IS MISPLACED - POSSIBLE ERROR
16	HALF-WORD ALIGNMENT HAS OCCURRED - POSSIBLE ERROR
17	FIELD n HAS A RELOCATABLE IN PLACE OF ABSOLUTE
18	FIELD n HAS AN ERROR IN LITERAL DEFINITION
19	DC SPECIFIED BUT NO VALUE LIST - POSSIBLE ERROR
20	FIELD n HAS UNUSED REGISTER SPECIFIED FOR DROP - POSSIBLE ERROR
21	FIELD n HAS A REGISTER EXPRESSION RELOCATABLE OR GREATER THAN 15
22	NO DBG CARD GENERATED - POSSIBLE ERROR
23	FIELD n HAS INVALID FORMAT SPECIFICATION - POSSIBLE ERROR

Code	Messages
24	FIELD n HAS AN INVALID LABEL - POSSIBLE ERROR
25	FIELD n HAS AN INVALID INTEGER - POSSIBLE ERROR
26	FIELD n HAS AN INVALID ADDRESS - POSSIBLE ERROR
27	FIELD n HAS AN INVALID CONDITION SPECIFICATION - POSSIBLE ERROR
28	FIELD n HAS AN ERROR IN REGISTER SPECIFICATION - POSSIBLE ERROR
29	(symbol) NOT USED
30	ADDRESS ON END CARD IN ERROR
31	FIELD n HAS AN INVALID EXPRESSION VALUE
32	NAME FIELD ON TITLE CARD INVALIDLY SPECIFIED - POSSIBLE ERROR
33	ADDRESS IN USING IS IVALID
34	FIELD n HAS AN INVALID REGISTER FOR USING
35	INVALID OPERATION CODE - NOP GENERATED
36	FIELD n HAS A NON-FLOATING POINT REGISTER SPECIFIED - POSSIBLE ERROR
37	FIELD n HAS A NON-EVEN REGISTER SPECIFIED - POSSIBLE ERROR
38	FIELD n HAS ADDRESS WHICH IS NOT COVERED BY A USING
39	FIELD n HAS ADDRESS WHICH MAY BE ERRONEOUSLY ALIGNED - POSSIBLE ERROR
40	FIELD n HAS ADDRESS FOR WHICH BOTH AN IMPLIED AND SPECIFIED REGISTER APPLY
41	FIELD n HAS SYMBOL WHOSE IMPLIED LENGTH IS TOO LARGE
42	FIELD n HAS A SHIFT AMOUNT WHICH IS LARGER THAN 64 - POSSIBLE ERROR
43	(symbol) CAUSED SYMBOL TABLE TO OVERFLOW

Code	Messages
44	FIELD n HAS DIVISION WHICH RESULTED IN ZERO QUOTIENT
45	RLD TABLE OVERFLOWED
46	(symbol) HAS NOT BEEN PREVIOUSLY DEFINED
47	ERROR IN MODIFIER(S)
48	FIELD n HAS TWO CONSECUTIVE QUOTES AFTER SYMBOL X
51	ERROR IN VALUE LIST
52	FIELD n USING REGISTER 0 AS A BASE - POSSIBLE ERROR
53	FIELD n ATTEMPT TO USE NON-ZERO VALUE FOR REGISTER ZERO
54	FIELD n HAS A VALUE WHICH IS TOO LARGE
55	SYSTEM ERROR
56	TRUNCATION OF CONSTANT - POSSIBLE ERROR
57	INCOMPLETE SCALING - POSSIBLE ERROR
58	FRACTION HAS BEEN OMITTED IN FLOAT. PT. NUMBER POSSIBLE ERROR
59	FLOATING-POINT CONSTANT IS TOO LARGE - POSSIBLE ERROR
60	CONSTANT HAS BEEN ROUNDED AND TRUNCATED - POSSIBLE ERROR
61	EXPONENT IS INVALID
62	FIELD n HAS ENTRY WHICH IS NOT IN PROGRAM OR COMMON
66	FIELD n HAS EXPRESSION WITH INVALID RELOCATABILITY
67	(symbol) SYMBOL SHOULD NOT APPEAR IN NAME FIELD
68	ATTEMPT TO DEFINE NEW CONTROL SECTION PREVIOUSLY DEFINED - POSSIBLE ERROR
69	(symbol) IS A DUPLICATE DEFINED ENTRY POINT - POSSIBLE ERROR

Code	Messages
70	(symbol) IS A DUPLICATE DEFINED EXTRN - POSSIBLE ERROR
71	START CARD MISSING - POSSIBLE ERROR
72	DOUBLE WORD ALIGNMENT HAS OCCURRED - POSSIBLE ERROR
73	ERROR IN DATA ITEM
74	CSECT TABLE HAS OVERFLOWED
75	LITERAL TABLE HAS OVERFLOWED
76	ENTRY TABLE HAS OVERFLOWED
77	LOCATION COUNTER HAS EXCEEDED MAXIMUM
78	(symbol) IS NOT DEFINED, PERHAPS BECAUSE OF SYMBOL TABLE OVERFLOW
79	LITERAL CANNOT BE REFERENCED BECAUSE OF SYMBOL TABLE OVERFLOW
80	FIELD n RESULTED IN A CONSTANT WHICH WAS TOO LARGE - POSSIBLE ERROR
81	FRACTION PART LOST - POSSIBLE ERROR
82	VALUE SPECIFICATION MISSING - POSSIBLE ERROR
83	FLOATING-POINT EXPONENT UNDERFLOW - POSSIBLE ERROR
84	VALUE EXCEEDS 24 BITS. RESULT WILL BE TRUNCATED
85	JOVIAL INPUT RECORDS MISSING - POSSIBLE TAPE ERROR

Comment on forced 'END' card: END CARD SUPPLIED BY ASSEMBLY*******

6.5 INPUT FORMAT

An assembly language source program consists of a sequence of statements punched into cards. An assembly language statement is composed of from one to four fields; starting from left to right, they are: name field, operation field, operand field, and comments field. The identification-sequence field (columns 73-80) is not part of the statement.

There are five general rules that must be observed when writing assembly language statements.

1. Every statement requires an operation field; additional fields are optional.

2. The fields in a statement must be in order, and they must be separated from one another by at least one blank, which acts as the field delimiter.

3. Because a blank is used as a delimiter, the name, operation, and operand fields must not contain embedded blanks. However, a blank may occur within a character self-defining value, a character constant, or a character literal.

4. Column 72 should always be blank.

5. If a card is completely blank (or if there is no operation field), it will be ignored by the assembly program.

In the various examples and statement formats throughout this publication, characters and words that may be written in assembly language statements are printed in capital letters. Some of these characters and words have special meaning to the assembly program (e.g., instruction mnemonics); others are representative examples of what might be written in statements.

Specifications for the various fields are presented in the following text.

6.5.1 Name Field

The name field is used to assign a symbolic name to a statement. Other statements can refer to a particular statement by its symbolic name. If a name is used, it must start in the begin column of the statement and it may occupy up to eight columns. The begin column is normally column 1, but it may be changed by the use of an ICTL assembly instruction. A name is always a symbol and must conform to the rules for symbols. The following example shows the symbol FIELD234 used as a name.

Name	Operation	Operand
FIELD234	DS	CL200

If the begin column is blank, the assembly program assumes that the statement has no name. The begin column is also used to indicate that a card is a comments card (see Subsection 6.5.4).

6.5.2 Operation Field

The operation field is used to specify a machine instruction or assembly instruction. This field may start in any column to the right of the begin column, provided that at least one blank separates it from the last character of the name. The operation field may contain any valid mnemonic operation code. A valid machine-instruction or assembly-instruction mnemonic cannot exceed five characters.

The following example shows the mnemonic code for the compare instruction (RR format) used in a statement named COMPARES.

Name	Operation	Operand
COMPARES	CR	5,6

6.5.3 Operand Field

The contents of the operand field provide the assembly program with information about the instruction specified in the operation field. If a machine instruction has been specified, the operand field specifies such program elements as registers, storage addresses, immediate data, masks, and storage-area lengths. For an assembly instruction, the operand field conveys whatever information the assembly program requires for the particular instruction.

The operand field may begin in any column to the right of the operation field, provided that at least one blank space separates it from the last character of the mnemonic code.

Depending on the instruction, the operand field may be composed of one or more subfields, called operands. Each operand must be separated from another by a comma. (Remember that a blank delimits the field; thus, blanks may not intervene between operands and commas.) The two operands in the following example specify general registers 5 and 6.

Name	Operation	Operand
COMPARES	CR	5,6

6.5.4 Comments Field

Comments are strictly for the convenience of the programmer. They permit lines of descriptive information about the program to be inserted into the program listing. Comments appear only in the program listing; they have no effect on the assembled object program. Any valid characters (including blanks) may be used as comments.

The comments field must appear to the right of the operand field; at least one blank must separate the comments from the last operand. An entire card can be used for comments by placing an asterisk in the begin column. If multiple lines of comments are desired, they must be written as separate comments cards with an asterisk in the begin column. See the example below.

Name	Operation	Operand	72
*THE ASTERISK IN COLUMN 1 MAKES THIS A COMMENTS CARD *THE ASTERISK IS REQUIRED IN EACH COMMENTS CARD COMPARES	 CR	 5,6 NO ASTERISK NEEDED	SHOULD BE BLANK

The programmer may use comments in instructions that do not require the operand field to be specified. In instructions where an optional operand field is omitted but a comments field is to be provided, the absence of the operand field must be indicated by a comma preceded and followed by one or more blanks. The next example illustrates this rule.

Name	Operation	Operand
	END	, THESE ARE COMMENTS

6.5.5 Identification-Sequence Field

The identification-sequence field is used for program identification and statement sequence numbers. This field occupies columns 73-80 of the input cards. If the field, or a portion of it, is used for program identification, the identification is punched in every statement card. The assembly program, however, does not normally check this field; it merely reproduces the information in the field on the output listing of the program.

If the identification-sequence field, or a portion of it, is used for statement sequence numbers, the numbers are punched in ascending sequence in successive input cards. By using the ISEQ assembly instruction, the programmer requests the assembly program to verify the ascending order of the numbers which he has punched.

6.6 OUTPUT LISTING FORMAT

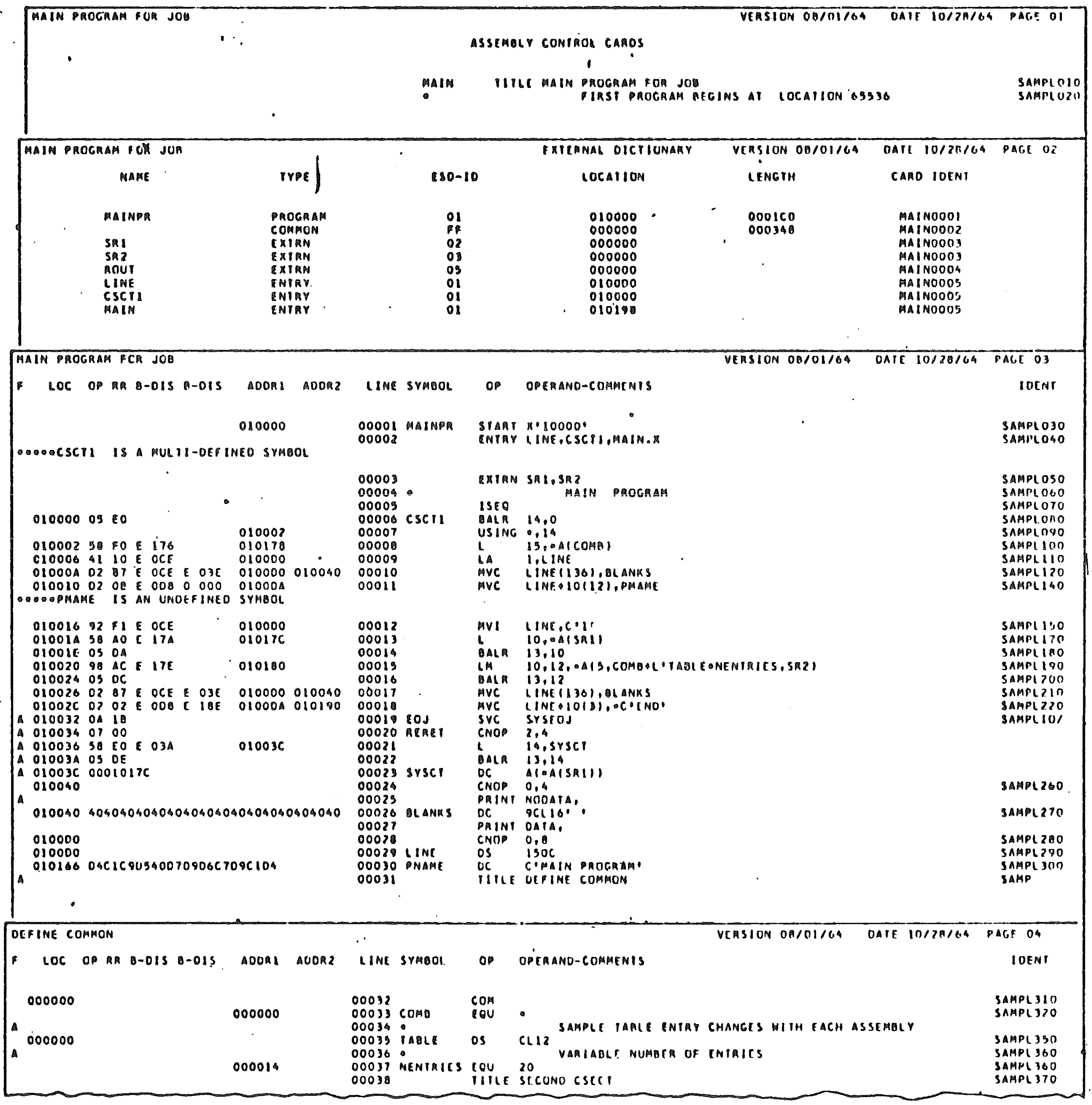
If the LIST option is specified on the $BAL (or $JOV) control card, a program listing is produced during Pass 2 of the assembly. Every statement in the program is printed as a separate line, unless the programmer makes use of the suppress option. The programmer may suppress the listing by omitting the LIST option from the $BAL (or $JOV) control card, or part of the listing may be suppressed by using the PRINT, SPEM (suppresses printing of possible error messages), or NLIST instructions in the BAL source program.

Figure 6-21 contains a sample of program listing. Page 01 of the listing contains the information given in the TITLE card and any comment cards following the TITLE card. Page 02 of the listing is an external symbol listing, containing program name, type, ESD-ID number, location in storage, length (in hexadecimal), and card identification. The actual program listing begins on page 03. Each line of the program listing contains the following fields.

Code	Field
F	Flag
LOC	Location
OP	Assembled output
RR	Assembled output
B-DIS	Assembled output
B-DIS	Assembled output
ADDR1	Effective address of operands 1 and 2
ADDR2	Effective address of operands 1 and 2
LINE	Line number
SYMBOL	Symbol (BAL)
OP	Operation code (BAL)
OPERAND-COMMENTS	Operand (BAL) and comments
IDENT	Identification sequence

The relocation dictionary follows the program listing, and contains the ESD-ID of the section where the address constant was defined (ID-LOC); the ESD-ID of the defined address constant (ID-DEF); the number of bytes of the address constant (LENGTH); the sign of the address constant (SIGN): and card identification (CARD IDENT). A cross-reference listing follows the relocation dictionary listing, and is, in turn, followed by a list of undefined symbols (if any) and a summary of errors.

The fields of the program listing are described in the following text.



```
MAIN PROGRAM FOR JOB                                                        VERSION 08/01/64   DATE 10/28/64   PAGE 01

                                           ASSEMBLY CONTROL CARDS

                                  MAIN     TITLE MAIN PROGRAM FOR JOB                                            SAMPL010
                                  *                          FIRST PROGRAM BEGINS AT  LOCATION 65536            SAMPL020
```

```
MAIN PROGRAM FOR JOB                                  EXTERNAL DICTIONARY   VERSION 08/01/64   DATE 10/28/64   PAGE 02

          NAME               TYPE            ESD-ID            LOCATION           LENGTH           CARD IDENT

         MAINPR             PROGRAM            01               010000            0001C0           MAIN0001
                            COMMON             FF               000000            000348           MAIN0002
         SR1                EXTRN              02               000000                             MAIN0003
         SR2                EXTRN              03               000000                             MAIN0003
         ROUT               EXTRN              05               000000                             MAIN0004
         LINE               ENTRY              01               010000                             MAIN0005
         CSCT1              ENTRY              01               010000                             MAIN0005
         MAIN               ENTRY              01               010198                             MAIN0005
```

```
MAIN PROGRAM FCR JOB                                                        VERSION 08/01/64   DATE 10/28/64   PAGE 03
F   LOC  OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP    OPERAND-COMMENTS                               IDENT

                             010000         00001 MAINPR   START X'10000'                                       SAMPL030
                                            00002          ENTRY LINE,CSCT1,MAIN.X                              SAMPL040
*****CSCT1  IS A MULTI-DEFINED SYMBOL

                                            00003          EXTRN SR1,SR2                                        SAMPL050
                                            00004 *                  MAIN  PROGRAM                              SAMPL060
                                            00005          ISEQ                                                 SAMPL070
  010000 05 E0                              00006 CSCT1    BALR  14,0                                           SAMPL080
                             010002         00007          USING *,14                                           SAMPL090
  010002 58 F0 E 176         010178         00008          L     15,=A(COMB)                                    SAMPL100
  010006 41 10 E 0CE         010000         00009          LA    1,LINE                                         SAMPL110
  01000A D2 87 E 0CE E 03E   010000 010040  00010          MVC   LINE(136),BLANKS                               SAMPL120
  010010 D2 0B E 0D8 0 000   01000A         00011          MVC   LINE+10(12),PMAME                              SAMPL140
*****PMAME  IS AN UNDEFINED SYMBOL

  010016 92 F1 E 0CE         010000         00012          MVI   LINE,C'1'                                      SAMPL150
  01001A 58 A0 E 17A         01017C         00013          L     10,=A(SR1)                                     SAMPL170
  01001E 05 DA                              00014          BALR  13,10                                          SAMPL180
  010020 98 AC E 17E         010180         00015          LM    10,12,=A(5,COMB+L'TABLE*NENTRIES,SR2)          SAMPL190
  010024 05 DC                              00016          BALR  13,12                                          SAMPL200
  010026 D2 87 E 0CE E 03E   010000 010040  00017          MVC   LINE(136),BLANKS                               SAMPL210
  01002C D2 02 E 0D8 E 18E   01000A 010190  00018          MVC   LINE+10(3),=C'END'                             SAMPL220
A 010032 0A 1B                              00019 EOJ      SVC   SYSEOJ                                         SAMPL10/
A 010034 07 00                              00020 RERET    CNOP  2,4
A 010036 58 E0 E 03A         01003C         00021          L     14,SYSCT
A 01003A 05 DE                              00022          BALR  13,14
A 01003C 0001017C                           00023 SYSCT    DC    A(=A(SR1))
  010040                                    00024          CNOP  0,4                                            SAMPL260
A                                           00025          PRINT NODATA,
  010040 40404040404040404040404040404040  00026 BLANKS   DC    9CL16' '                                       SAMPL270
                                            00027          PRINT DATA,
  010000                                    00028          CNOP  0,8                                            SAMPL280
  010000                                    00029 LINE     DS    150C                                           SAMPL290
  010166 D4C1C9D5400709D6C709C1D4          00030 PNAME    DC    C'MAIN PROGRAM'                                SAMPL300
A                                           00031          TITLE DEFINE COMMON                                  SAMP
```

```
DEFINE COMMON                                                               VERSION 08/01/64   DATE 10/28/64   PAGE 04
F   LOC  OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP    OPERAND-COMMENTS                               IDENT

  000000                                    00032          COM                                                  SAMPL310
                             000000         00033 COMB     EQU   *                                              SAMPL320
A                                           00034 *                  SAMPLE TABLE ENTRY CHANGES WITH EACH ASSEMBLY
  000000                                    00035 TABLE    DS    CL12                                           SAMPL350
A                                           00036 *                  VARIABLE NUMBER OF ENTRIES                 SAMPL360
                             000014         00037 NENTRIES EQU   20                                             SAMPL360
                                            00038          TITLE SECOND CSECT                                   SAMPL370
```

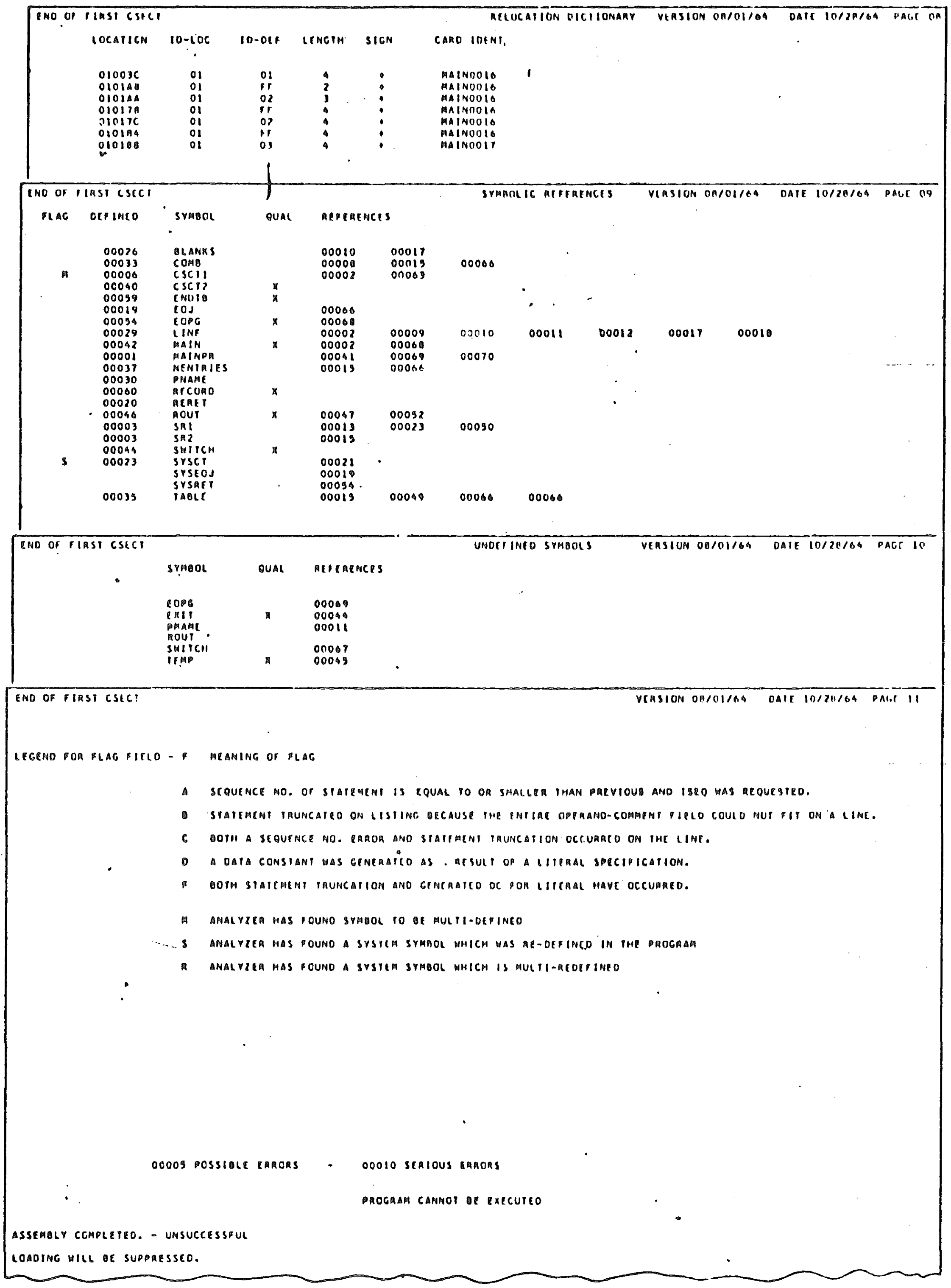
FIGURE 6-21. SAMPLE PROGRAM LISTING (SHEET 1 OF 3)

```
SECOND CSECT                                                                           VERSION 08/01/64   DATE 10/20/64  PAGE 05

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2  LINE SYMBOL    OP    OPERAND-COMMENTS                                          IDENT


                                            00039          QUAL  X                                                         SAMPL380
  010198                                    00040 CSCT2    CSECT                                                           SAMPL390
                                            00041          ORG   MAINPR.+X'2000'                                           SAMPL400
*****FIELD 1 HAS EXPRESSION WITH INVALID RELOCATABILITY

  010198 05 C0                              00042 MAIN     BALR  12,0                                                      SAMPL410
                              01019A        00043          USING *,12                                                      SAMPL420
  01019A 47 00 0 000                        00044 SWITCH   NOP   EXIT                                                      SAMPL430
*****EXIT  IS AN UNDEFINED SYMBOL

  01019E 90 2B 0 000                        00045          STM   2,11,TEMP                                                 SAMPL440
*****TEMP  IS AN UNDEFINED SYMBOL

                                            00046          EXTRN ROUT                                                      SAMPL450
  0101A2 58 D0 E 18A          01018C        00047          L     13,=A(ROUT)                                               SAMPL460
  0101A6 05 ED                              00048          BALR  14,13                                                     SAMPL470
  0101A8 0000                               00049          DC    AL2(TABLE.)                                               SAMPL480
  0101AA 000008                             00050          DC    AL3(SR1.+8)                                               SAMPL490
  0101AD 0000002A                           00051          DC    FL4'42'                                                   SAMPL500
  0101B1 00
*****HALF WORD ALIGNMENT HAS OCCURRED - POSSIBLE ERROR

  0101B2 58 00 0 000                        00052          L     13,=(ROUT)                                                SAMPL510
*****FIELD 2 HAS AN ERROR IN LITERAL DEFINITION

  0101B6 05 ED                              00053          BALR  14,13                                                     SAMPL520
  0101B8 0A 19                              00054 EOPG     SVC   SYSRET.                                                   SAMPL530
                                            00055          SSEQ                                                            SAMP
                                            00056          TITLE ADD TO COMMON DECLARATION                                 SAMPL
```

```
ADD TO COMMON DECLARATION                                                              VERSION 08/01/64   DATE 10/28/64  PAGE 06

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2  LINE SYMBOL    OP    OPERAND-COMMENTS                                          IDENT


  C0000C                                    00057          COM                                                             SAMPL
  C0000C                                    00058          DS    200F                                                      SAMP
  00032C                                    00059 ENDTB    DS    F                                                         SAMPL
  000330 40                                 00060 RECORD   DC    20C' '                                                    SAMPL
  000331 40
  000332 40
  000333 40
  000334 40
  000335 40
  000336 40
  000337 40
  000338 40
  000339 40
  00033A 40
  00033B 40
  00033C 40
  00033D 40
  00033E 40
  00033F 40
  000340 40
  000341 40
  000342 40
  000343 40
                                            00061          TITLE END OF FIRST CSECT                                        SAMPL
```

```
END OF FIRST CSECT                                                                     VERSION 08/01/64   DATE 10/28/64  PAGE 07

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2  LINE SYMBOL    OP    OPERAND-COMMENTS                                          IDENT


                                            00062          QUAL                                                            SAMPL
  010172                                    00063 CSCT1    CSECT                                                           SAMP
*****CSCT1  IS A MULTI-DEFINED SYMBOL

                                            00064 *                DEFINE CONSTANTS                                        SAMP
  010172                                    00065          LTORG                                                           SAMP
  010172 0000CC00C000
D 010178 00C00000                                          DC    A(COMB)
D 01017C 00000C00                                          DC    A(SR1)
D 010180 00000C05CC0000F000000000                          DC    A(5,COMB+L'TABLE*NENTRIES,SR2)
D 01018C 0C000C00                                          DC    A(ROUT)
*****ROUT  IS AN UNDEFINED SYMBOL

D 010190 C5D5C4                                            DC    C'END'


              ON CARD-ID MAIN0013           00066 EOJ      DUMP  ALPH,COMMON,COMB,TABLE+L'TABLE*NENTRIES                   SAMP
                                            00067 SWITCH   DUMPC HEX,CONDMP,X'10000,X'15000',5,10 '                        SAMP
*****FIELD 3 HAS INVALID PUNCTUATION
*****FIELD 3 HAS AN INVALID ADDRESS - POSSIBLE ERROR
*****NO DBG CARD GENERATED - POSSIBLE ERROR

              ON CARD-ID MAIN0014           00068          TRACE LOOPTR,MAIN.X,EOPG.X                                      SAMP
                                            00069          DUMPE HEXI,EMERG,MAINPP,EOPG                                    SAMP
*****EOPG  IS AN UNDEFINED SYMBOL
*****FIELD 4 HAS AN INVALID ADDRESS - POSSIBLE ERROR.
*****NO DBG CARD GENERATED - POSSIBLE ERROR


  C10193                      010000        00070          END   MAINPR                                                    SAMP
```

FIGURE 6-21. SAMPLE PROGRAM LISTING (SHEET 2 OF 3)



```
END OF FIRST CSECT                                     RELOCATION DICTIONARY   VERSION 08/01/64   DATE 10/28/64   PAGE 08

      LOCATION    ID-LOC     ID-DEF    LENGTH   SIGN      CARD IDENT,

       01003C       01         01        4        +        MAIN0016
       0101A8       01         FF        2        +        MAIN0016
       0101AA       01         02        3        +        MAIN0016
       010178       01         FF        4        +        MAIN0016
       01017C       01         07        4        +        MAIN0016
       010184       01         FF        4        +        MAIN0016
       010188       01         03        4        +        MAIN0017
```

```
END OF FIRST CSECT                                     SYMBOLIC REFERENCES     VERSION 08/01/64   DATE 10/28/64   PAGE 09

  FLAG   DEFINED     SYMBOL        QUAL    REFERENCES

          00026      BLANKS                00010     00017
          00033      COMB                  00008     00015     00066
   M      00006      CSCT1                 00002     00063
          00040      CSCT2           X
          00059      ENDTB           X
          00019      EOJ                   00066
          00054      EOPG            X     00068
          00029      LINE                  00002     00009     00010     00011     00012     00017     00018
          00042      MAIN            X     00002     00068
          00001      MAINPR                00041     00069     00070
          00037      NENTRIES              00015     00066
          00030      PNAME
          00060      RECORD          X
          00020      RERET
          00046      ROUT            X     00047     00052
          00003      SR1                   00013     00023     00050
          00003      SR2                   00015
          00044      SWITCH          X
   S      00023      SYSCT                 00021
                     SYSEOJ                00019
                     SYSRET                00054
          00035      TABLE                 00015     00049     00066     00066
```

```
END OF FIRST CSECT                                     UNDEFINED SYMBOLS       VERSION 08/01/64   DATE 10/28/64   PAGE 10

                     SYMBOL        QUAL    REFERENCES

                     EOPG                  00069
                     EXIT            X     00044
                     PNAME                 00011
                     ROUT
                     SWITCH                00067
                     TEMP            X     00045
```

```
END OF FIRST CSECT                                                             VERSION 08/01/64   DATE 10/28/64   PAGE 11


LEGEND FOR FLAG FIELD - F   MEANING OF FLAG

                        A   SEQUENCE NO. OF STATEMENT IS EQUAL TO OR SMALLER THAN PREVIOUS AND ISEQ WAS REQUESTED.

                        B   STATEMENT TRUNCATED ON LISTING BECAUSE THE ENTIRE OPERAND-COMMENT FIELD COULD NOT FIT ON A LINE.

                        C   BOTH A SEQUENCE NO. ERROR AND STATEMENT TRUNCATION OCCURRED ON THE LINE.

                        D   A DATA CONSTANT WAS GENERATED AS . RESULT OF A LITERAL SPECIFICATION.

                        E   BOTH STATEMENT TRUNCATION AND GENERATED DC FOR LITERAL HAVE OCCURRED.

                        M   ANALYZER HAS FOUND SYMBOL TO BE MULTI-DEFINED

                        S   ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH WAS RE-DEFINED IN THE PROGRAM

                        R   ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH IS MULTI-REDEFINED




                  00005 POSSIBLE ERRORS    -    00010 SERIOUS ERRORS

                                                PROGRAM CANNOT BE EXECUTED

ASSEMBLY COMPLETED. - UNSUCCESSFUL
LOADING WILL BE SUPPRESSED.
```

FIGURE 6-21. SAMPLE PROGRAM LISTING (SHEET 3 OF 3)

6.6.1 F Field

The F field is usually blank, but may contain one of the following alphabetic characters if the specified condition exists (these flags apply to the program listing and do not affect assembly of the program).

Flag	Conditions
A	Indicates an error in sequence numbers (sequencing is checked only if an ISEQ request is made in the BAL source program).
B	Indicates an excessively long operand field (will not fit on the line).
C	Indicates that both a sequencing error and an excessively long operand field occur in the same statement.
D	Indicates a constant that has been generated by a literal (printing of literals may be suppressed by the PRINT instruction in the BAL source program).
F	Indicates that both an excessively long operand field and a constant generated by a literal occur in the same statement.
M	Indicates that the analyzer found a multi-defined symbol.
S	Indicates that the analyzer found a system symbol which was redefined in the problem program. This flag is for information only; it does not signal an error.
R	Indicates that the analyzer found a system symbol which was multi-redefined

Whenever an entry appears in the flag field, the assembler automatically prints a legend for the flags at the end of the listing. See page 11 of the listing in Figure 6-21.

6.6.2 LOC Field

The LOC field contains a six-character hexadecimal representation of the address assigned to the first byte of the object code produced for the statement.

6.6.3 Assembled Output

The assembled output contains the object code produced for the statement. This field is divided into four subfields.

Subfield	Contents
OP	A two-character OP code in hexadecimal.
RR	A two-character subfield containing register and length information.
B-DIS	The first base register and displacement (blank if none).
B-DIS	The second base register and displacement (blank if none).

6.6.4 OPERAND 1 and 2 Addresses

Subfield	Contents
ADDR1	Effective address of the first operand (blank if none).
ADDR2	Effective address of the second operand (blank if none).

For constants, 16 bytes are printed on each line. If the constant requires more than 16 bytes, additional lines are used to print it (unless a PRINT statement in the BAL source program has suppressed the additional lines). Constants that have a multiplicity greater than one require multiple lines for printing.

6.6.5 LINE Field

The LINE field contains a number assigned to each input statement by the assembler. This line number is used by the symbolic analyzer for identifying the location of symbol definitions and references.

6.6.6 SYMBOL Field

The SYMBOL field contains the symbol appearing in the same field of the BAL source statement. If there is none, the field remains blank.

6.6.7 OP Field

The OP field contains the operation code appearing in the operation field of the BAL source statement.

6.6.8 OPERAND-COMMENTS Field

The OPERAND-COMMENTS field contains the operand and contents of the comments field specified in the BAL source statement.

6.6.9 IDENT Field

The IDENT field contains the contents of columns 73-80 of the BAL source statement, which are used by the SPT edit program or for card identification and sequencing.

6.6.10 Cross-Reference Listing of Symbols

When the ANALYZ option on the $BAL (or $JOV) control card is specified, the symbolic analyzer is called to produce a cross-reference listing of all symbols used in a program. This option forces generation of a program listing. The NLISTed statements are not processed by the BAL analyzer. Consequently, any multi-defined symbols, system symbol redefinitions, etc. which occur in NLISTed code will not be detected by the analyzer and the appropriate flags will not appear in the listing. The cross-reference listing entitled ''Symbolic References,'' appears after the relocation dictionary listing.

The symbolic analyzer is a part of the assembler, and the use of this option requires at least one additional pass to produce the cross-reference listing. The listing gives the line number of each symbol definition and the line numbers of all statements that refer to the symbol. Multi-defined symbols and redefined system symbols are flagged. A separate listing of any undefined symbols is printed after the symbol listing. The symbolic analyzer can handle any number of references, but if more than 255 undefined symbols occur, only the first 255 are printed and a diagnostic message, stating that there are unlisted and undefined symbols, will be printed.

6.6.11 Diagnostics

The assembler prints a diagnostic message in the program listing for errors discovered during processing of a BAL source program. The diagnostic message(s) is printed immediately after the erroneous statement. If no LIST option was present on the $BAL card, or if an NLIST is in effect for the erroneous statement, the assembler will force printing of the statement and all unsuppressed diagnostics. The asterisks that precede each message are for ease of identification in the program listing.

6.7 OUTPUT CARD FORMAT

If the PUNCH option is used on the $BAL control card, an object deck is produced by the assembler. The object deck begins with a $OBJ control card and is followed by the other cards described below.

The assembler may produce seven types of cards in the object deck from the BAL source program: Text (TXT), External Symbol Dictionary (ESD), Relocation List Dictionary (RLD), End (END), Debug (DBG), Library (LIB), and Object ($OBJ). The purpose and format of these cards are described in the following text.

6.7.1 $OBJ Control Card

The $OBJ control card is produced by the assembler and placed at the beginning of the object deck. Any object deck used as input to the loader must be preceded by this card.

6.7.2 ESD Card

The 9020 loader permits separately assembled programs to be relocated, loaded, and executed together. These assembled programs can be referred to each other and to common storage. The ESD card makes it possible for one program to refer to symbols defined in another program. The ESD card contains the information necessary for the loader to relocate the program and to assign addresses that were unknown at assembly time.

There are four types of entries on an ESD card. More than one type of entry can appear on a single card; each is identified, and different information is supplied by the assembler for each type. A maximum of three entries can appear on one ESD card.

The following list shows the four types of entries and describes the pertinent information that is filled in by the assembler.

1. Program Name - The program name ESD card has the same name as that supplied by the START card in the compiler or assembler source deck. There will always be a program name ESD card produced

by the assembler; if no program name was given, the assembler assigns the name of .NONAME. The program name is always the first ESD entry, and is assigned an identification number of 01 (ESD-ID). In addition to the program name and ESD-ID number, the ESD card included a code that identified the entry as a program, the starting address assigned by the assembler, and the number of bytes in the program. There is only one program name card for each assembly. The program name ESD may include a code that indicates that the program is an overlay. If so, the original or parent program must have the same name and precede the overlay in the order of loading.

2. Common Declaration - If the assembly (not produced by JOVIAL source code) included a source BAL COM card, an entry indicating a common declaration is made on an ESD card. The information entered for common storage includes an identifying code, a starting address of zero, and the number of bytes in common storage. Only one common declaration may be made for a program, and the ESD entry is assigned an arbitrary ESD-ID of 255. Common storage may be used by every subprogram in the same job load which has a common declaration.

3. External Symbols - External symbols are symbols (addresses) which are not defined in the same program that refers to them. They are made known to the assembler by means of the EXTRN pseudo-operation. Each external symbol, as it is discovered by the assembler, is assigned a sequential ESD-ID number from 02 to 254 (01 is reserved for the program name and 255 for common storage; separate CSECTs not defined by the starting address of the program are assigned ESD-ID numbers but do not of themselves produce ESD cards of any type). This number is punched in the ESD card, together with the symbol, and an address of zero. To conserve space, only the first ESD-ID is punched on the card. The loader adds 1 to the first ESD-ID to make the second ESD-ID, etc.

4. Entry Points - Entry points are symbols (addresses) that are defined in one program and can be referred to by EXTRNs in another program. They are made known to the assembler by the ENTRY pseudo-operation. The entry symbol, a code identifying it as an entry point, and the address at which it was assembled are punched in the ESD card. No ESD-ID numbers are given to entry points.

Note: On an ESD card, all fields are hexadecimal, unless otherwise noted, and appear in EBCDIC card code. See the IBM System/360 Reference Data Card, X20-1703-5.

The format of an ESD card is as follows.

Column	Content
1	12-2-9 multiple punch (identifies this as a card acceptable to the loader).
2-4	ESD (identifies type of load card).
5-8	Blank.
9-10	Checksum. (If two blanks, checksum is ignored.)
11-12	Number of bytes of text to be loaded from card: if card contains one symbol, 16 (12-0-1-8-9 and 12-11-1-8-9 multiple punches; in hex, 0010); if card contains two symbols, 32 (12-0-1-8-9 and 11-0-1-8-9 multiple punches; in hex, 0020); if card contains three symbols, 48 (12-0-1-8-9 and 12-11-0-1-8-9 multiple punches; in hex, 0030).
13-14	Blank.
15-16	ESD-ID number of program name (01 or in hex, 0001) or first external symbol on card (02-254 in hex, 0002 - 00FE) (blank, if all symbols are entry points).
17-24	Name of first symbol on card (1-8 alphameric characters in EBCDIC, left-justified). Blank if common declaration.
25	Number identifying type of symbol: 00 (12-0-1-8-9 multiple punch, if program name); 01 (12-2-9 multiple punch, if external symbol); 0A (12-2-8-9 multiple punch, if common declaration); 0B (12-3-8-9 multiple punch, if entry point in common storage).

Column	Content
26-28	If program name, assembler-assigned address of the first byte of the program. If entry point, assembler-assigned address of entry point. If external symbol or common storage: zeros (12-0-1-8-9 multiple punches in all three columns)
29	Blank.
30-32	If program name, number of bytes in assembled program. If entry point, ESD-ID. If external symbol, blank If common declaration, number of bytes in common storage.

Note: Columns 33-71 are blank if only one symbol is on the card. Columns 33-48 are used for the second symbol; columns 49-64 are used for the third symbol.

Column	Content
33-40 (49-56)	Name of second (third) symbol on card (1-8 alphameric characters, in EBCDIC, left-justified, blank if common declaration).
41 (57)	Number identifying type of symbol: 01 (12-1-9 multiple punch, if entry point. 02 (12-2-9 multiple punch, if common declaration) 0A (12-2-8-9 multiple punch, if common declaration) 0B (12-2-8-9 multiple punch, if entry point in common storage)
42-44 (58-60)	If entry point: assembler-assigned address of the entry point. If external symbol or common storage, zeros (12-0-1-8-9 multiple punches in all three columns).

Column	Content
45 (61)	Blank.
46-48 (62-64)	If entry point or external symbol blank. If common declaration, number of bytes in common storage.
65-71	Blank.
72	If name, of original program, blank. If name of overlay program, any numeric punch.
73-80	Card identification.

6.7.3 TXT Card

The TXT cards contain the actual text of the object program. Each gives the address of the first byte of text to be loaded from the card, the number of bytes that the card holds, and the text itself. The last of these is a variable field that may contain up to 56 bytes of information in EBCDIC.

Note: On a TXT card, all fields are hexadecimal, unless otherwise noted, and appear in EBCDIC card code. See the IBM System/360 Reference Data Card, X20-1703-5.

The TXT card format is as follows.

Column	Content
1	12-2-9 punch (identifies this as a card acceptable to the loader).
2-4	TXT (identifies type of load card).
5	Blank.
6-8	24-bit assembled address of first byte of variable field to be loaded from card.
9-10	Checksum. (If two blanks, checksum is ignored.)
11-12	Number of bytes of information in the variable field. (Columns 17-72)

Column	Content
13-14	Blank.
15-16	01 or 255 - ESD-ID number (255 is assigned by the assembler to text in common storage). In hex, 0001 or 00FF.
17-72	Variable field (up to 56 bytes of object program text in EBCDIC to be loaded).
73-80	Card identification.

6.7.4 RLD Card

Most relocatable addresses can be automatically relocated by changing the contents of the base register (the displacement remains constant). RLD cards must be provided to enable the loader to relocate address constants. These cards indicate to the loader those address constants that will have to be changed if the program is relocated. There are three variables: the assembler address of the address constant; the location of this assembled address (whether it is in the program or in common storage), called the position header; and the identity of the symbol whose relocated address must be filled in, known as the relocation header. The postion header will be 01, if the address constant is in the program, or 255, if it is in common storage. The relocation header will be 01, if the symbol is defined within the program. If the symbol is defined by an EXTRN, the relocation header will be the identifying ESD-ID number assigned to the symbol by the assembler. If the symbol is defined in common storage, the relocation header will be 255.

Each RLD entry on the card identifies four items: the relocation header, position header, a flag, and the address of the constant. The flag is coded to specify: (a) the length (one to four bytes) of the address constant, (b) whether to add or subtract the relocation factor, and (c) a continuation code. The continuation code informs the loader that either the next constant (following the current constant) on the card has a new relocation header, position header, flag, and address; or that it has the same relocation header and position header as the previous one, and therefore consists only of a flag and address.

Consider the following example.

```
PROG1          START          X '800'
               ENTRY          PROG1A
                                 .
                                 .
                                 .
PROG1A         LR             14,15
                                 .
                                 .
                                 .
ADCON1         DC             A(PROG1A)
ADCON2         DC             A(PROG2)
ADCON3         DC             A(PROG2+8)
ADCON4         DC             A(8-PROG2)
ADCON5         DC             A(8-PROG1A)
               END
```

Assume that PROG1 is assembled at hexadecimal location 800. PROG1 will have an ESD-ID of 01, since it is a program name. PROG1A is an entry point and is assumed to be assembled at location 880. PROG2 is an external symbol and has an ESD-ID of 02. Under these conditions, the assembler will assign the address constant ADCON1 a value of 880 and the address constant ADCON2 a value of 0, since ADCON2 contains the address of the external symbol. The assembler will assign both ADCON3 and ADCON4 the displacement value +8, since the value of the external symbol is assumed to be zero during assembly. ADCON5 would be assigned a 4-byte negative value of the difference of 8-880 (hecadecimal 878). In the RLD, the postion header for all address constants will be 01. The relocation headers will be 01 for ADCON1 and ADCON5; 02 for ADCON2, ADCON3, and ADCON4. At load time, if PROG1 were loaded at 900 (the relocation constant thus being +100) and PROG2 were loaded at 2100, the value of ADCON1 would be increased by +100 to become 980. The value of ADCON2 would be +2100; ADCON3 would be +2108; ADCON4 would be -20F8; ADCON5 would be -978.

Note: On an RLD card, all fields are hexadecimal, unless otherwise noted, and appear in EBCDIC card code. See the IBM System/360 Reference Data Card, X20-1703-5.

Column	Content
1	12-2-9 multiple punch (identifies this as a card acceptable to the loader).
2-4	RLD (identifies the type of loader card).
5-8	Blank.
9-10	Checksum. (If two blanks, checksum is ignored.)
11-12	Number of bytes of information that follow.
17-18	Relocation header: ESD-ID number of the symbol referred to in the DC statement if defined by an EXTRN; 01 if defined within the program.
19-20	Position header: 01 if address constant is in the program; 255 if in common storage. (In hex, 0001 or 00FF.)
21	Flag field: indicates the length of the address constant, whether the relocation constant should be added to or subtracted from the address constant, and whether another address constant follows the current one for this RLD entry. (Tables 6-1 and 6-2 show the information to be punched in this column depending on whether the continuation flag indicates the same or a new relocation header/position header combination for the next item, respectively.)
22-24	Address of the address constant.

Note: If needed, 4-column fields in the same format as columns 21-24 may be subsequently repeated in columns 25 and following provided they have the same relocation header/position header combination. The format matches that of column 21-24 and the continuation flag for all of the 4-column fields except the last will be a Table 6-1 type flag. When a Table 6-2 type flag is found at any point, it means that either an 8-column field, in the format of columns 17-24 will be found next beyond the current field or that this is the last field on the particular RLD card.

TABLE 6-1. RLD-FLAG FIELD-SAME HEADER

Length of Address Constant In Bytes	Relocation Constant To Be Added To Address Constant		Relocation Constant To Be Subtracted From Address Constant	
	EBCDIC Card Punch	Hexadecimal Equivalent	EBCDIC Card Punch	Hexadecimal Equivalent
1 Byte	12-1-9	01	12-3-9	03
2 Bytes	12-5-9	05	12-7-9	07
3 Bytes	12-1-8-9	09	12-3-8-9	0B
4 Bytes	12-5-8-9	0D	12-7-8-9	0F

TABLE 6-2. RLD-FLAG FIELD-NEW HEADER

Length Of Address Constant In Bytes	Relocation Constant To Be Added To Address Constant		Relocation Constant To Be Subtracted From Address Constant	
	EBCDIC Card Punch	Hexadecimal Equivalent	EBCDIC Card Punch	Hexadecimal Equivalent
1 Byte	12-0-1-8-9	00	12-2-9	02
2 Bytes	12-4-9	04	12-6-9	06
3 Bytes	12-8-9	08	12-2-8-9	0A
4 Bytes	12-4-8-9	0C	12-6-8-9	0E

Example: A given RLD card has a Table 6-1 flag in column 21. This means that a 4-column field begins in column 25. Upon examining column 25, a Table 6-2 flag is found. The address in columns 26-28 has the same header combination as that in columns 22-24, but an 8-column field begins in column 29 and the continuation flag will be in column 33. Suppose column 33 has a Table 6-1 type flag. This means that a 4-column field will begin in column 37. Now suppose that the field from column 37-40 is the last field on the card. Therefore the flag in column 37 must contain a Table 6-2 type flag.

6.7.5 END Card

The END card marks the conclusion of any object deck. The END card is prepared by the assembler from the information entered on the source program END statement. It may also designate the first executable instruction of the job by referring to the symbolic name or the assembled address of an instruction. If the symbol in the END statement is defined by an EXTRN, the assembler punches the alphanumeric symbol on the loader END card (Type 2); otherwise, the assembled address is punched on the loader END card (Type 1).

The three formats for the END card are listed in the following text.

Note: On an END card, all fields are hexadecimal, unless otherwise noted, and appear in EBCDIC card code. See the IBM System/360 Reference Data Card, X20-1703-5.

6.7.5.1 Type 1 END Card. The following format is used when the END card designates the hexadecimal address of the first executable instruction.

Column	Content
1	12-2-9 punch (identifies this as a card acceptable to the loader).
2-4	END (identifies type of load card).
5	Blank
6-8	24-bit assembled address of first executable instruction. (It must occur within the program terminated by the END card. It need not appear in the ESD.)
9-14	Blank

Column	Content
15-16	ESD-ID number (must be 01 or 255).
17-22	Blank.
73-80	Card identification.

6.7.5.2 Type 2 END Card. The following format is used where the END card designates the symbolic name of the first executable instruction.

Column	Content
1	12-2-9 punch (identifies this as a card acceptable to the loader).
2-4	END (identifies type of load card).
5-16	Blank.
17-24	Symbolic name of first executable instruction (alphameric, left-justified within the field). The symbolic name must be defined on an ESD card as an entry point or program name somewhere in the job load. It need not be defined in the particular object deck that the END card terminates. For example, an END card terminating PROG1 may designate PROG2 or an entry point in PROG2 providing PROG2 is within the same job load.
25-72	Blank.
73-80	Card identification.

6.7.5.3 Type 3 END Card. The following format is used if the END card does not designate a hexadecimal address or a symbolic name as first executable instruction.

Column	Content
1	12-2-9 punch (identifies this as a card acceptable to the loader).
2-4	END (identifies type of load card).
5-72	Blank.
73-80	Card identification.

6.7.6 DBG Card

The DBG card requests execution-time debugging, and is produced from the programmer's symbolic (BAL) debugging request. As prepared for inclusion at assembly time, the debugging requests are coded in BAL (described in the publication IBM 9020 Data Processing System: Debugging System User's Manual (DEBUGG) and are translated by the assembler into loader language.

6.7.7 LIB Card

The LIB cards are used to place compiled JOVIAL programs and/or routines on the library tape. If the compiler output contains a statement which specifies that the program or routine is to be placed on the library tape, the assembler produces the LIB card. The LIB card is further described in the publication Library Edit User's Manual.

6.7.8 XREF Deck

Three types of cards may be produced by the assembler in the XREF punched deck: an XRF header card (.XRF3), an XRF symbol card (.XRF4), and an XRF trailer card (.XRF7). The purpose of these cards is described in the following text; the formats are described in the Subprogram Design for the Compool Reference Matrix Program (XREF).

6.7.8.1 .XRF3 Card. The XRF header card contains the name of the program for which the XRF deck was punched. It informs the XREF subprogram that an XRF deck follows.

6.7.8.2 .XRF4 Card. The XRF symbol card contains the program name and up to eight compool data names and/or library routine names referenced by the assembled program.

6.7.8.3 .XRF7 Card. The XRF trailer card contains the program name and the count of the number of compool data names and library routine names referenced by the assembled program. This card informs the XREF Subprogram that the XRF deck has been completed.

6.8 COMPOOL FORMAT

The assembler reads the ''Reserves'' portion of a compool as contained on a standard compool or MLC tape. See the Utility NOSS Monitor User's Manual (UTILITY) for the format of a compool or MLC tape. See the Subprogram Design Document (SDD) Compool Edit Program (CMPEDT) for the format of the compool file on these tapes.

7.0 RESTRICTIONS, LIMITATIONS, AND ASSUMPTIONS

The BAL assembler, as herein documented, is designed to operate as a System Processor under the NOSS Monitor. It interfaces with the NOSS Monitor, and assumes that the minimum NOSS hardware configuration, in addition to anything detailed in this SDD, will be available.

The BAL Assembler is also designed to interface with the JOVIAL Compiler and the JOVIAL Compool.