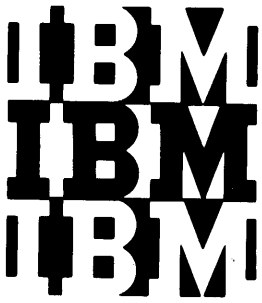


Y26-3598-0

Program Logic

IBM SYSTEM/360 OPERATING SYSTEM
ASSEMBLER (32K)
PROGRAM LOGIC MANUAL

This publication describes the internal logic of the 32K Assembler for IBM System/360 Operating System. It is intended for use by persons involved in program maintenance, and system programmers who are altering the program design. Program Logic information is not necessary for the use and operation of the program; therefore, distribution of this publication is limited to those with the aforementioned requirements.

PREFACE

PREREQUISITE AND RELATED LITERATURE

Effective use of this Program Logic Manual (PLM) requires a thorough understanding of the contents of the publication IBM Operating System/360, Assembler Language (Form C28-6514).

DESIGN OF THIS PUBLICATION

This PLM supplements the program listing. Labels on flowcharts are keyed to the program listing.

This manual consists of an introduction, two sections containing detailed information about the various phases, and a section of reference material.

The Introduction describes the purpose of the program, its relationship to the Operating System, the I/O devices required to perform the program, and the program organization.

The phase descriptions describe the phase logic and functions performed by the phase. The labels in the 'Routine Description' portion of each phase are arranged alphabetically, assuming that the user will be coming to the descriptions from a flowchart, looking for an explanation of a label encountered in one of the flowcharts.

The reference section of the manual consists of flowcharts, appendixes, and a glossary of terms.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. A form has been provided at the back of this publication for reader's comments. If the form has been detached, comments may be directed to: IBM Corporation, Programming Publications, Department 232, San Jose, California 95114.

ILLUSTRATIONS

PURPOSE OF THE ASSEMBLER

IBM System/360 Operating System consists of a control program and a number of processing programs (Figure 1). The control program governs the order in which the processing programs are executed and provides services that are required in common by the processing programs during their execution. The processing programs consist of language translators and service programs that are provided by IBM to assist the user of the system, as well as problem programs that are written by the user and incorporated as part of the system.

The assembler translates a source program, coded in IBM System/360 Operating System Assembler Language, into a relocatable machine-language object program, assigns storage locations to instructions and other elements of the program, and performs auxiliary assembler functions designated by the programmer. Several optional outputs are available, including a printed listing of the source and object statements, TESTRAN cross reference listing, and additional information useful to the programmer in analyzing his program, such as error indications.

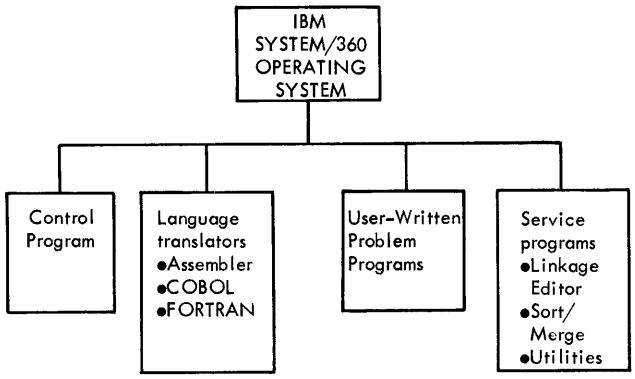


Figure 1. General Organization of the Operating System

SYSTEM ENVIRONMENT

The Assembler operates as a processor program under OS/360 on System/360 Models 30, 40, 50, 65, and 75, with at least 32, 768 bytes of main storage.

The minimum System/360 I/O requirements are as follows (see Figure 2):

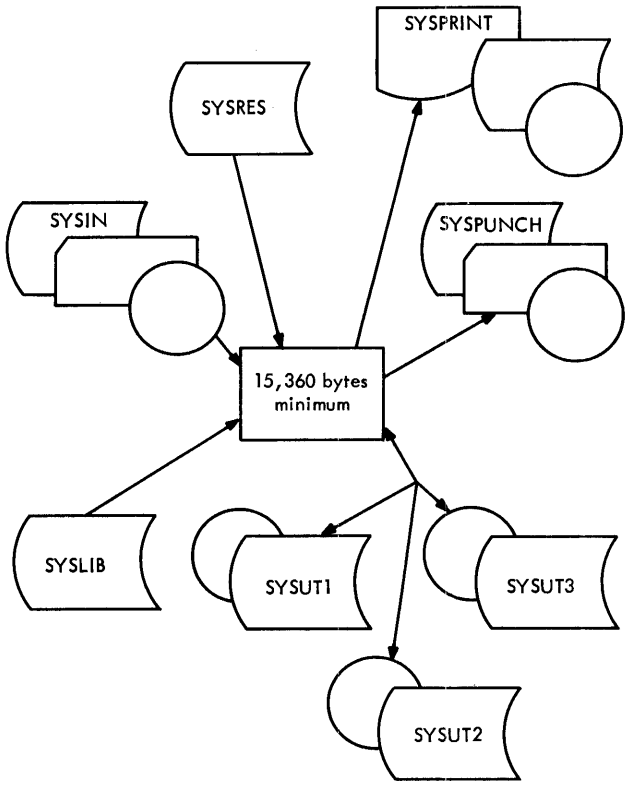


Figure 2. Input/Output Requirements

- Minimum requirements of the OS/360 Control Program
- At least 15,360 bytes of contiguous main storage available to Assembler
- Data sets having the following DD names:
  SYSRES - System Residence data set containing the Assembler Program, the OS/360 Control Program, etc. SYSRES must be DASD resident.
  SYSIN (BSAM) data set containing source statement text for assembly. SYSIN may be DASD, magnetic tape, or card resident. The format is card images of 80 bytes per logical record, unblocked.
  SYSLIB - Assembler Source Library (BPAM) data set containing system macro definitions, and source text which may be COPYed into the main text of the program or into programmer and system macro definitions.
  SYSPRINT (BSAM) data set on which the assembler will place output text for printing; SYSPRINT may be printer, magnetic tape, or DASD resident. The format is 121-byte logical records, unblocked.
  SYSPUNCH (BSAM) data set. The output text produced (relocatable object program) may be included in one data set with the DECK or LOAD option. Separate DECK and LOAD data sets are not supported. The format is 81 bytes per logical record (80 on card), unblocked. SYSPUNCH output may be on cards, DASD, or magnetic tape.
  SYSUT1, SYSUT2, SYSUT3 (BSAM)) Utility data sets used for external storage; the use and formats of these data sets will vary from one phase to another. All text, certain tables, and table segments reside on these data sets which may be DASD, magnetic tape, or mixed.

NOTE: Standard labels are required on disk-resident data sets; standard labels or no labels may be used with magnetic tape resident data sets. User labels are not supported.

USE OF ADDITIONAL FACILITIES

Main Storage

Additional main storage, if available, will be used to expand certain tables.

Control Program Services

The Assembler uses the control program services listed in Appendix L. The data sets and assembler phases using these services are shown in Appendix M.

PROGRAM ORGANIZATION

Major Components

The Assembler consists of the following major components (throughout this manual the IET portion of each phase name is not used when referring to the various phases):

Formal Name	Abbreviated Name*
IETASM (Resident)	AFF
IETMAC	ACF
IETINP	DCB
IETE1	1FF
IETE2/E2A	2FF/2AF
IETE3/E3A	3FF/3AF
IETE4P/E4M/E4S	40F/4FF/4AF
IETE5P/E5/E5A/E5E	50F/5FF/5AF/5EF
IETRTA	6AF
IET07/07A/07B	7FF
IET07I	7DF
IET08/08A/08B	8FF
IETRTB	6BF
IET09	9FF
IET09I	9DF
IET10	10F
IET10B	10B
IET21A	21A
IET21B/21C/21D	21B/21C/21D
IETPP	CFF
IETDI	DFF

*To assist in identifying core dumps, the first three bytes of each phase contain an abbreviated hexadecimal phase name and the phase model and level as follows.

X'E21B31'

Digit	Significance
1	E for Assembler E (32K)
2-4	Abbreviated Phase Name
5	Model
6	Level

Program Flow - Figure 3

The Assembler performs two major functions in processing source programs: macro generation and conditional assembly, and symbolic to machine language assembly.

At the beginning of the program, the resident portion of the Assembler is read into main storage. This portion of the assembler contains I/O tables and other information required by the macro generator and assembly portions. The resident portion passes control to MAC which in turn passes

control to Phase E1 to begin macro generation.

The generator portion (Phases E1 through E5) reads in copy code, performs conditional assembly, and expands macro-instructions into one-for-one statements. The last macro generator phase returns control to MAC which passes control to RTA. RTA calls the first assembly phase (07).

The assembly phases (07 through DI) convert one-for-one statements into machine language instructions and constants, produce a relocatable object program, listings and cross-references, and print diagnostics. The last assembly phase returns control to RTB which passes control to RTA which, in turn, passes control back to resident phase ASM.

Figure 4 illustrates the I/O flow for each phase of the program.

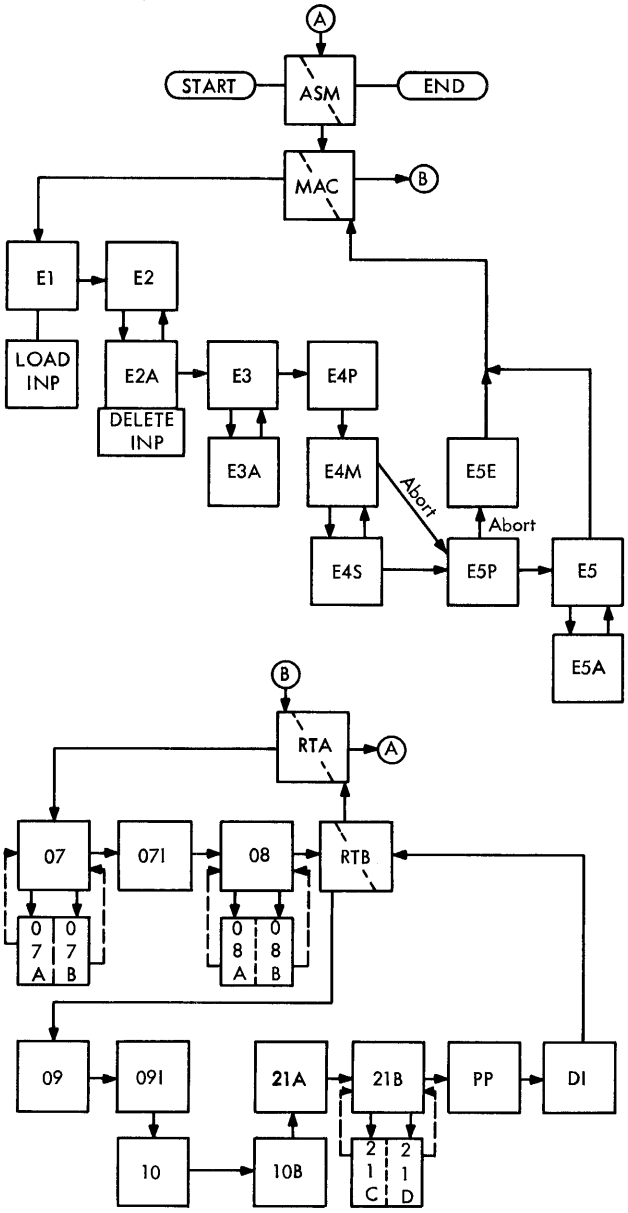


Figure 3. Program Flow

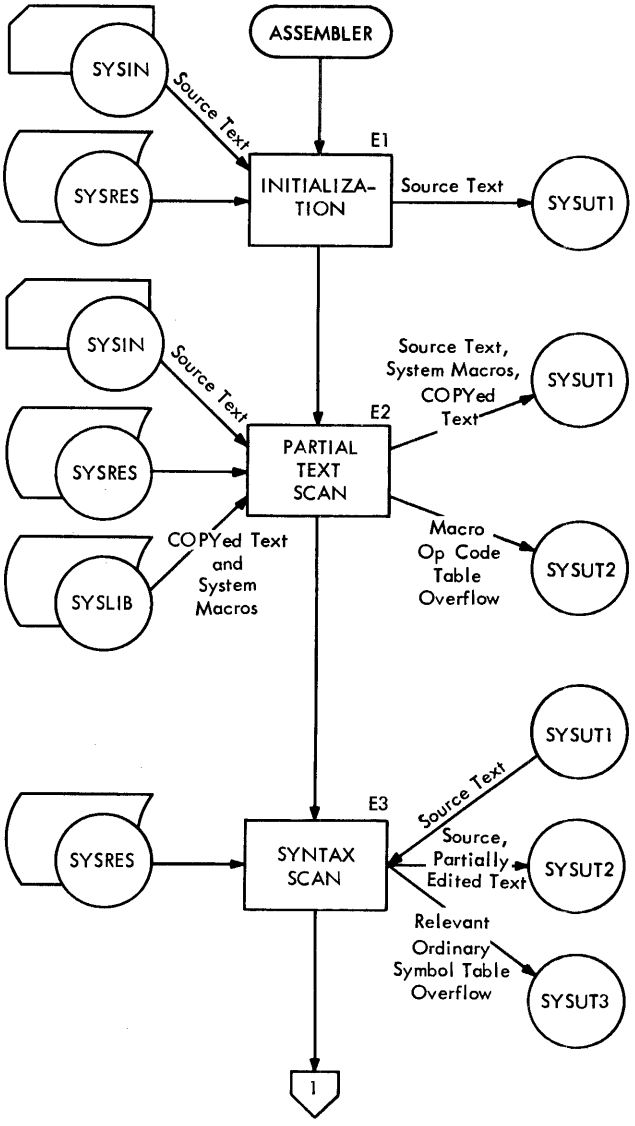


Figure 4. Input/Output Flow (Part 1 of 5)

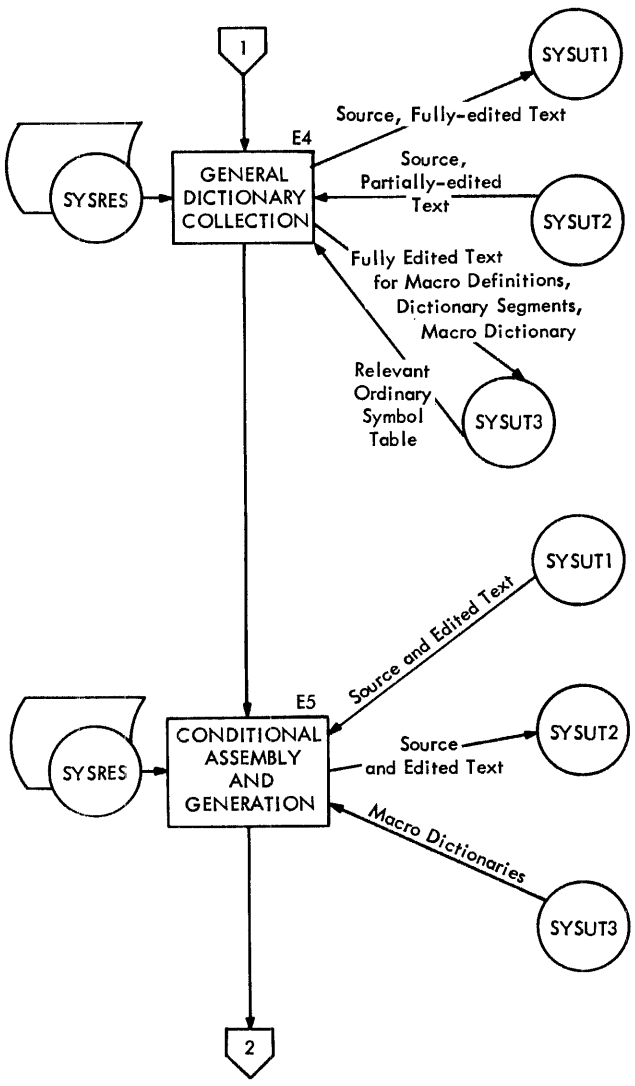


Figure 4. Input/Output Flow (Part 2 of 5)

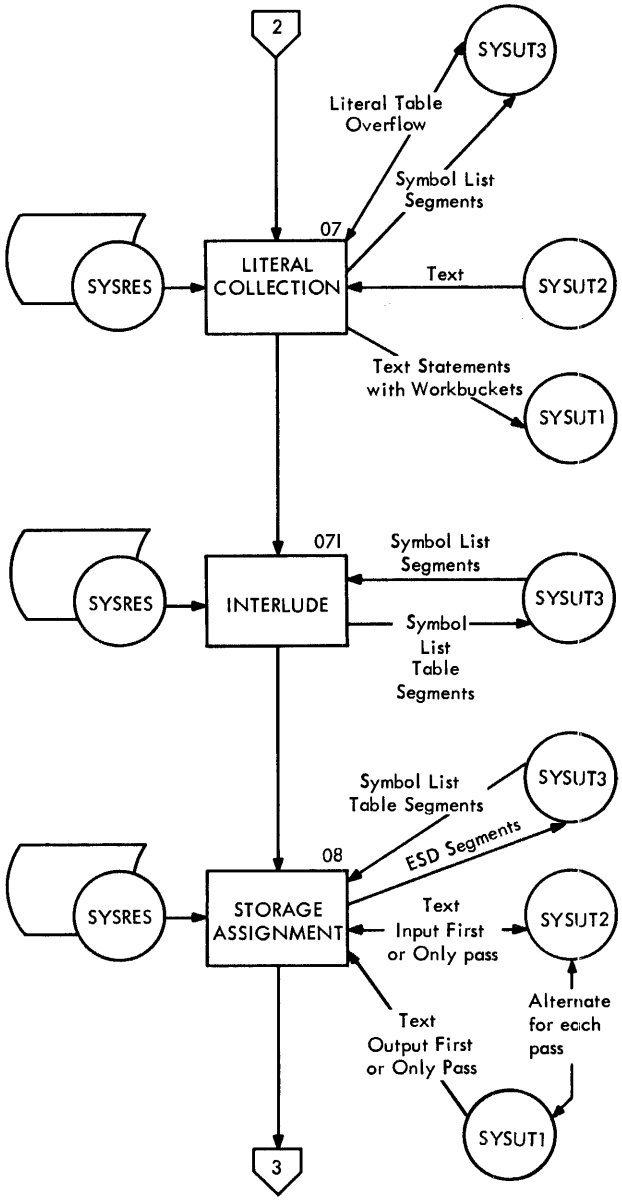


Figure 4. Input/Output Flow (Part 3 of 5)

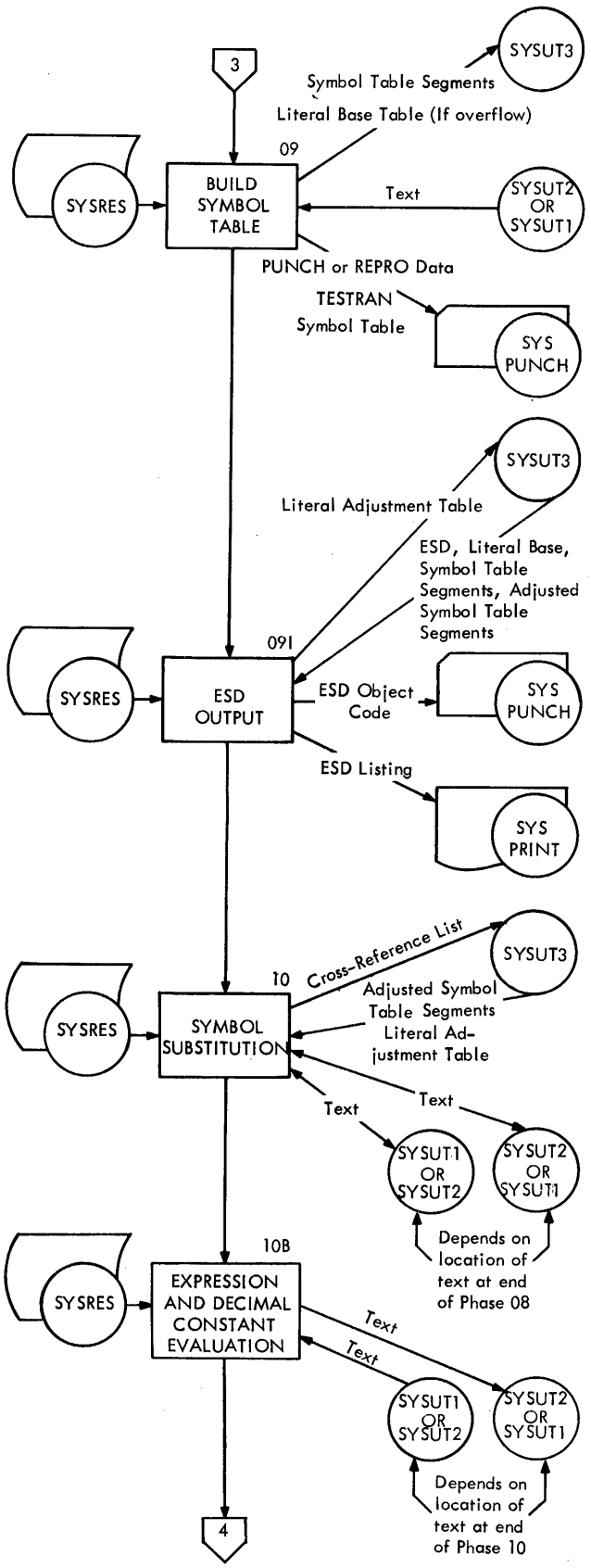


Figure 4. Input/Output Flow (Part 4 of 5)

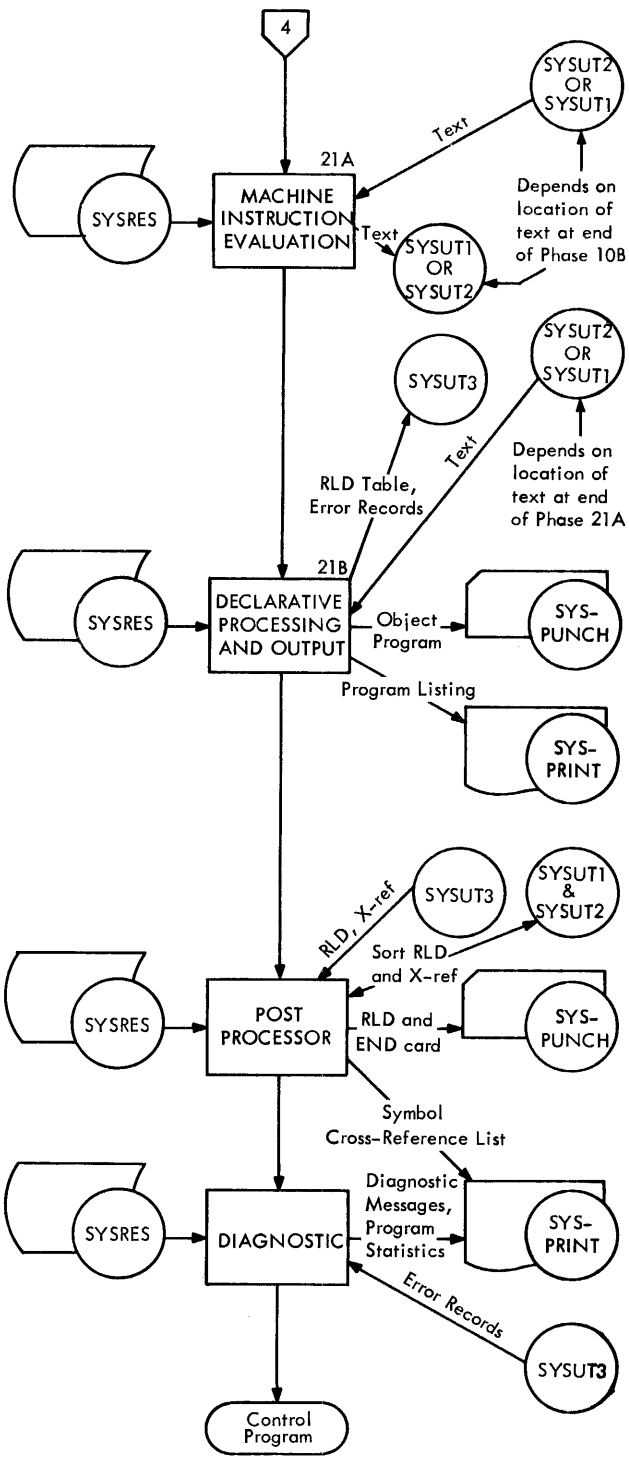


Figure 4. Input/Output Flow (Part 5 of 5)

Macro Generation and Conditional Assembly. This portion of the assembler requires five phases (with overlays) and makes four passes over the text. The first pass inputs all statements and partially scans the text to construct logical statements from one or more source statements. The second pass performs a syntax scan of each statement and produces a partially-edited text for input to the third pass. The third pass forms a global and one or more local dictionaries of various symbols and macro mnemonic operation codes. The fourth pass performs the actual macro generation and conditional assembly.

Assembly Phases. The assembly portion (Phases 07 through DI) receives edited text from the generator, determines the amount of main storage to be reserved for each statement, assigns locations, and completes the symbolic assembly process. Seven or more passes are required, additional passes being required when the Symbol Table and/or Symbol List Table overflow.

Functions of Major Components

Program Segment ASM (Master Root Segment). Resident program segment. Contains DCBs and DECBs for SYSUT1, SYSUT2 and SYSUT3.

Program Segment MAC (Macro Generator I/O). This segment contains the I/O routines used by the macro generator phases.

Program Segment INP (Input DCBs and DECBs). This segment contains the DCBs and DECBs for SYSIN and SYSLIB.

Phase E1 (Initialization). Phase E1 performs initialization for the macro generator, and determines the operating environment by interrogating the Control Program to determine the storage available, I/O devices, and I/O channel configurations. Phase E1 loads INP, opens the utility data sets, input data set, and library data set, processes parameters from the EXEC card, and finally processes the ICTL statement, if present.

Phase E2 (Partial Text Scan). Phase E2 reads source statements from the input data set, system macro definitions and COPYed text from the library data set. The source statements are written along with logical text records and any error records. The logical text is input to Phase E3. Some syntactic errors are detected and error records written as necessary. Operation codes are translated and appended to the logical text statements with other control information.

Phase E3/E3A (Syntax Scan). Phase E3 scans logical statements to determine their syntax and produces partially edited text. Some syntactic errors are detected and flagged for subsequent error processing.

Phase E4P/E4M/E4S (Dictionary Collection). Phase E4 reads the partially-edited text produced in Phase E3/E3A, and uses this text to create dictionaries containing macro mnemonics, sequence symbols, variable symbols, and relevant ordinary symbols for each macro defined in the program, for the main portion of the program, and for global information defined in the program. These dictionaries are used to insert pointers to Phase 5 dictionaries, thus producing fully edited text.

Phase E5P/E5/E5A/E5E (Conditional Assembly and Generation). Using the dictionaries and fully edited text produced in Phase E4, Phase E5 completes the dictionary entries, fully edits the macro definitions, generates one-for-one statements from the macro definitions and performs conditional assembly. The output from Phase E5 serves as input to Phase 07, the first assembly phase.

Root Segment RTA. Contains the system utility unit I/O routines for the assembly phases.

Phase 07/07A/07B (Literal Collection). Phase 07 collects and inserts all literals into the program stream at the appropriate positions, converts all self-defining numbers to their binary values, translates all substituted mnemonic operation codes created during macro generation, and builds Symbol List Table segments for use by Phase 08. This table contains symbols encountered in duplication factor and length modifier expressions, scale or exponent modifier expressions, EXTRN, EQU, ORG, AND CNOP operand fields and in CSECT, DSECT, and START name fields. If the Symbol List segment exceeds the assigned area, it overflows onto SYSUT3.

Phase E7I (Interlude). Phase 07I reformats short Symbol List segments into Symbol List Table sections for Phase 08.

Phase 08/08A/08B (Storage Assignment). Phase 08 evaluates all expressions requiring previous definition of symbols, assigns relative storage addresses to all instructions, constants, and storage areas, and creates External Symbol Dictionary (ESD) segments.

Root Segment RTB. Contains the DCBs and DECBs for SYSPRINT and SYSPUNCH.

Phase 09 (Build Symbol Table). Phase 09 enters all symbols found in the name fields of the program into Symbol Table segments, processes the TESTRAN Symbol Table (optional), and any PUNCH or REPRO statements which precede the first control section. It also creates Literal Base (location) Table segments.

Phase 09I (ESD Output). Phase 09I produces the External Symbol Dictionary (ESD) and writes it out on the appropriate data sets (SYSPRINT and SYSPUNCH). This phase also adjusts the address values of all symbols in the Symbol Table segments to reflect the order of CSECTs within the assembly and creates Literal Adjustment Table segments.

Phase 10 (Symbol Substitution). Phase 10 substitutes storage addresses for the symbols used in the operand field of all instructions, using the Symbol Table and Literal Adjustment Table segments built in Phase 09.

Phase 10B (Expression and Decimal Constant Evaluation). Phase 10B evaluates expressions not requiring previous definition and stores the results in the appropriate text record. USING, DROP, and listing control instructions are processed. USING and listing control records are inserted into the text stream for use by Phases 21A and 21B.

Phase 21A (Machine Instruction Evaluation). Phase 21A reformats storage addresses to base-displacement format and writes them as partially formatted text for Phase 21B. Type D, E, F, and H constants are converted to their binary equivalents.

Phase 21B/21C/21D (Declarative Processing and Output). This phase converts, and evaluates, if necessary, declarative statements (e.g., DC/DS) to their object form and formats them for listing. This phase also creates Relocation List Dictionary (RLD) Table segments for relocatable constants and V-type constants.

Phase PP (Post Processor). The Post Processor phase writes the Relocation List Dictionary (RLD) and Loader END card on SYSPUNCH, and writes the Symbol Cross-Reference Table on SYSPRINT if this option is requested.

Phase DI (Diagnostic). The Diagnostic phase writes diagnostic messages and program statistics on SYSPRINT.

MACRO GENERATION AND CONDITIONAL ASSEMBLY PHASES

The macro generation and conditional assembly portion requires five phases (with overlays) and makes four passes over the text. The first pass inputs all statements and partially scans the text to construct logical statements from the one or more card source statements. If COPY statements or system macro instructions are detected, text is included from the source library at the appropriate points. The second pass performs a syntax scan of each statement and produces a partially-edited text for input to the third pass. The third pass collects variable symbols, sequence symbols, macro mnemonic operation codes, and ordinary symbols whose attributes are required for conditional assembly or macro generation (relevant ordinary symbols) and forms a global dictionary and one or more local dictionaries. One local dictionary is required for the main portion of the program for relevant ordinary symbols, sequence symbols, and local variable symbols. One local dictionary containing sequence symbols and local variable symbols is required for each programmer or systems macro used.

The fourth pass performs the actual macro generation and conditional assembly.

PROGRAM SEGMENT ASM - MASTER ROOT SEGMENT

ASM contains the DCBs and DECBs for the system utility units SYSUT1, SYSUT2, and SYSUT3. The routine labeled MRS performs the initialization and contains the link (LINK macro) to MAC.

ASM, once loaded by the invoking program, remains in core throughout the assembly process.

PROGRAM SEGMENT MAC - MACRO GENERATOR I/O PACKAGE

MAC consists of the I/O routines used by the macro generator phases. These routines perform the following functions: READ, WRITE, NOTE, POINT TO READ, POINT TO WRITE, and CHECK. The units to which these functions apply are the system utilities, the system input unit (SYSIN) and the system library unit (SYSLIB).

The routine labeled MACGEN performs initialization and contains the linkage to ASM, Phase E1, and RTA.

MAC, once loaded by ASM, is resident in core only during the macro generation phases of the Assembler. When the macro generation phases are completed, control passes (RETURN macro) to MAC which, in turn, transfers (XCTL macro) to RTA.

PROGRAM SEGMENT INP - INPUT DCBs AND DECBs

INP consists of the DCBs and DECBs for the system units SYSIN and SYSLIB. INP is loaded (LOAD macro) by Phase E1. INP is deleted (DELETE macro) by Phase 2A of the macro generator.

PHASE E1 - INITIALIZATION AND ASSIGNMENT - CHART 01

Phase E1 performs initialization for the macro generator and determines the operating environment by interrogating the Control Program to determine the storage available. Phase E1 opens the utility data sets, input data set, and the library data set, processes the parameters of the EXEC card, and, finally, processes the ICTL statement, if it is present.

Functions

The functions of this phase are designed to perform the following:

- Initialize for macro generation.
- Process the EXEC parameters.
- Process the ICTL parameters, if present.

Phase E1 begins the macro generator initialization by obtaining core for COMMON, using the GETMAIN system macro instruction, and initializing it. The variable unconditional mode (vu) of GETMAIN is used throughout the Assembler to obtain core for COMMON, certain tables, and I/O buffers.

The system utility files SYSUT1, SYSUT2, and SYSUT3 are initialized and opened. INP, a table containing DCBs and DECBs pertaining to SYSIN and SYSLIB is then brought into core using the LOAD macro instruction. Phase E1 initializes and opens these files.

From the system input unit, SYSIN, this

phase reads the EXEC record. Its parameters are extracted and corresponding indicators are set within the COMMON area. (See IBM System/360 Operating System, Programmer's Guide, Form C28-6595 for a list and description of these parameters.) The line count is extracted, if present, converted to binary, and likewise stored in COMMON.

Phase E1 then reads the first source record. If this record is not an ICTL record, Phase E1 terminates and calls Phase E2. If the first source record is an ICTL record, its parameters are extracted and stored in the COMMON area. Phase E1 then terminates and calls Phase E2.

See Appendix D for illustrations and descriptions of the various record formats involved in this phase.

Routine Descriptions

BWFORC/BWRITE

This subroutine writes blocked text on units SYSUT1 and SYSUT2, including source text, edited text, error records, and macro and ordinary symbol references.

E1GO

This routine performs the initialization of COMMON, I/O indicators, etc., for Phase E1.

FINSCN

This routine tests the first record of the source text to determine if it is an ICTL record.

GETSRC

This subroutine reads the source statements, checking for continuation records, comments records, error records, the last record, record sequence, and statement format.

ISICTL

This routine processes and stores the operands found in the ICTL record.

NOINV

This routine loads the DCBs for SYSIN and SYSLIB, opens these files and the system utility files.

TUNEXT

This routine executes the transfer to Phase E2.

PHASE E2/E2A - PARTIAL TEXT SCAN - CHARTS 02 AND 03

Phase E2 reads source statements from the input data set, and system macro definitions and COPYed text from the library data set according to the setting of SYSIN. The source statements are written on SYS001 for inclusion on SYSPRINT. A partial syntactical scan is made on every statement primarily for mnemonic operation code translation and determination of unknown mnemonic operations (possible system macro). A table of unknown mnemonic operations is built and segments of this table are written on SYSUT2. The partially edited text of each statement is written on SYSUT1. After all source statements from the input data set have been read and processed, control is transferred to overlay E2A for determination of system macro input. Unknown mnemonic operations which are found to be system macros are then processed as further input to phase E2. The location on SYSLIB of the system macro source is obtained using the FIND macro.

Overlay E2A reads the Macro Name Table segments built in E2, building one complete table which has only one entry for each unique mnemonic. This table is scanned and all referenced and defined macros are flagged to prevent repetitive look up of the same macro. These macros (Subset A) are assumed to be either IBM or user system macros and are written on SYSUT2.

Any remaining undefined macros are collected (Subset B) and written on SYSUT2 to be looked up by Phase E2. Phase E2 is reloaded and SYSLIB becomes the system input unit.

When no undefined macro mnemonics remain in the Macro Name Table, control is transferred to Phase E3.

Functions

The functions of this phase are designed to perform the following:

- Construct logical text statements.
- Obtain System Macro Definitions and COPYed text from SYSLIB.

E2 reads the remainder of the source text from SYSIN, checks it for correct sequence according to any ISEQ statements present, and writes it as source records on SYSUT1 following the source text, if any, written in Phase E1. If no END record is found, one is created upon receiving an EOD indication from the Operating System. If

an ICTL statement is found, an END record is created and inserted into the output stream and the input is ended.

A Macro definition is considered to begin with a MACRO header statement and end with the next MEND trailer statement. All MACRO statements in between are considered invalid statements. If the MEND trailer for the last valid MACRO header has not been read before an END statement or EOD indication, one is generated at that time.

Logical source statements, which may be contained in one record or a record and one or more continuation records, are collected into a single logical statement record and interspersed with the source text on SYSUT1. Macro instructions and macro prototype statements are exceptional in that one logical statement record is created for each source record. If the operation is a machine instruction or assembler instruction, it is encoded and appended to the logical statement with other control information. If a COPY assembler instruction is detected, the text is located in the system library with FIND, is edited, and then inserted into the text stream being written on SYSUT1. If the text is not located, an error record is written on SYSUT1.

Overlay E2 builds a table of Macro operation codes, called the Macro Name Table, by assuming that undefined mnemonic operation codes are macro instructions. When entered into the table, flag bits are set indicating use or definition. A MACRO operation code associated with a macro prototype statement is flagged as defined. All other entries are flagged as used.

The Subset A is read from SYSUT2, SYSLIB is searched using FIND and System Macros are read from the library, scanned, and written on SYSUT1 at the end of the previous output. If a Macro cannot be located on the system library, the Macro Mnemonic is assumed to be an undefined operation and in error. If the MEND trailer label for a System Macro definition is not read before SYSLIB EOD is sensed, one is created at that time.

If macro instructions are nested in System Macro definitions, these Macro mnemonics are entered in Macro Name Table segments (as described earlier) and written on SYSUT2 following Subsets A and B. Phase E2A is then called again.

If Overlay E2A is called more than once, it reads in the Subset A, Subset B, and Macro Name Table segments and produces new subsetted tables, overwriting on SYSUT2 any previous output. E2 and E2A are loaded as many times as necessary to subset the Macro Name Table (E2A) and to locate the required macros (E2) until all macro names are indicated as defined.

See Appendixes D and I for illustrations and descriptions of the various record and table formats involved in this phase.

Routine Descriptions

MACLIB

This routine indicates to overlay E2 that SYSLIB has become the input unit. It also locates the undefined macro in the System Library.

DRIVER

This routine checks statement syntax, scans the name and operation fields, and determines if the operation is a normal symbol, machine operation, pseudo-operation, or undefined macro.

WRTAPE

Writes source on SYSUT3.

NOPSK

Sets type of edited text.

OPOK

Test Operation field for normal symbol.

MCRINS

This routine outputs the undefined macros onto SYSUT2.

BWFORC

See Phase E1.

NDSMT3

This routine causes a branch to BWFORC for writing on SYSUT1.

MEND

Processes the MEND statement.

END

Processes the END statement.

SY200

This routine sets SWTCH5 to indicate system macro editing.

DCLOSE

Common closeout routine for macro definitions and mainline program.

PREGET

This routine reads in the Macro Name

Table segments from SYSUT2.

MOWR

This routine performs the subsetting of the Macro Name Table, separating the defined and undefined macro names, and outputs the undefined macro table onto SYSUT2.

FINALE

This routine checks for macros that have not yet been defined.

WROT

This routine outputs the defined Macro Name Table onto SYSUT2 when the end of input is reached and returns to overlay E2.

NOMAC

This routine terminates Phase E2/E2A when all macros have been defined, and transfers from E2A to Phase E3.

PHASE E3/E3A - SYNTAX SCAN - CHARTS 04 AND 05

Phase E3 reads the source logical statement and error text from SYSUT1. If the statement is not a macro instruction or prototype, Phase E3 proceeds to scan for syntactical errors and creates a string language, partially-edited text and associated dictionary collection records. Relevant ordinary symbols are written on SYSUT3 and all other text on SYSUT2. This is continued until all input text has been processed, at which point control is passed to Phase E4P. Overlay E3A, when called, processes only macro instructions and prototype statements. E3A creates partially-edited text records and special pointer and operand dictionary collection records. Similarly relevant ordinary symbols are written on SYSUT3 and all other text on SYSUT2. Upon the completion of the macro instruction or prototype processing control is returned to E3.

Functions

The functions of the phase are designed to perform the following:

- Scan the logical statement text for syntactical errors.
- Produce a string-language, partially-edited text and associated dictionary collection records.
- Build the Relevant Ordinary Symbol Table.

The macro/conditional assembly generation dictionaries will contain entries for macro mnemonics, variable symbols, sequence symbols, and those ordinary symbols which appear in macro-instruction operands or whose attributes are referred to in conditional assembly statements.

Phase E3 creates special pointer and operand dictionary collection records but no partially-edited text records for global and local set symbol declarations (GBLx and LCLx). These special dictionary collection records will be used in the following dictionary collection phase to make definition entries for the associated set symbols in the appropriate dictionary. Phase E3 also collects, in standard pointer and operand dictionary collection records, all references to pertinent variable, sequence, and pertinent ordinary symbols in macro definition model statements, conditional assembly statements, and open code statements, i.e., statements outside of macro definitions. If a symbol names a DC/DS statement, Phase E3A collects the symbol and evaluates and collects the type, length, and scale attributes of the symbol. These standard dictionary collection records follow their associated partially-edited text records and will be used to assist in the processing of the partially-edited text records by the dictionary collection phase.

For macro instructions, there is one partially-edited text record with its associated standard dictionary collection records to describe both the name and operation fields. In addition, there is one partially-edited text record with its associated dictionary collection records to describe each macro instruction operand.

For prototype statements, all positional symbolic parameters are collected in special pointer and operand dictionary collection records that have no associated partially-edited text records preceding them.

In addition, there is a partially-edited text record with its associated standard dictionary collection records to describe each keyword operand.

Phase E3/E3A also collects all ordinary symbols relevant to conditional assembly into Relevant Ordinary Symbol Table segments which are written on SYSUT3. This table will be used by the succeeding dictionary collection phase to select those ordinary symbols and their attributes which are to be placed in local dictionaries.

The source text, partially-edited text, and error records, if any, are written on SYSUT2.

See Appendix D for illustrations and descriptions of the various record formats involved in this phase.

Routine Descriptions

ASCAN

This subroutine collects the attribute of DC/DS operands containing no variable symbols.

BEGINZ

Initialization for Phase E3.

DC/DS

Processes DC and DS statements.

DCLOSE

Closeout routine for macro-definition and mainline program.

DCLRTN

This routine processes global and local declarations, building special text records for each.

DRIVER

Initializes for a new statement.

DRV20

Scans the name field.

DRV30

Space over text to operation field.

DRV40

This routine checks for a macro instruction or a prototype statement. Overlay E3A is loaded to process these two types.

END

Processes the END statement.

ENDOPR

Processes the end of an operand or statement.

ENDSMT

Edits comments.

GETSRC

Reads source and macro statements.

GSCAN

Scans input text except operand fields significant to the macro generator calling sequence.

INITLZ

This routine initializes overlay E3, setting up the I/O buffer areas, common area, etc.

LEGOP

Performs a syntax scan of each operand parameter.

LOOKUP

Produces output pointer.

MCHINS

Processes machine instructions.

MCRINS

This routine processes the macro instruction.

MCRO5

Processes the output header for edited text records.

MEND

Processes the MEND statement.

MISCAN

Processes and scans macro-instructions.

NDOPRO

This routine writes edited text on SYSUT2.

NDSMT3

Writes output text.

NDSMT4

Skips records looking for MEND.

NOERR

Restores output pointer for start of next record.

NTBORC

Processes prototype parameters.

NXTOPO

Initializes for prototype scan.

NXTOP1

Starts scan of new operand within prototype statement.

NXTOP2

Inserts positional flag and edits the operand.

PROSCN

This routine edits the operand of a prototype statement.

PROTO

Produces operand pointer records for prototype statements.

PRO8

Processes prototype positional parameter.

PRSCNO

Edits prototype statement.

PSDOPR

Processes non-machine operation.

PTSBHD

Processes sublists and keyword operands.

RPUPFD

Inserts end of statement symbol into test.

RPUPOP

Inserts end of operand symbol into text.

SUBNX1

Initializes for sublist operand.

VALDOP

Tests for machine operation or Assembler instruction.

VARSYM

Recursive subroutine to scan variable symbols.

PHASE E4P/E4M/E4S - DICTIONARY COLLECTION - CHARTS 06, 07, 08, 09

Phase E4P initializes the COMMON area, I/O buffer, and dictionaries for Phase E4M.

Phase E4M reads source records, partially-edited text, and dictionary collection records from SYSUT2. Source records are immediately written on SYSUT1. When the start of open code is detected, the relevant ordinary symbols are read from SYSUT3 and entered into the General Dictionary.

At the start of a macro definition, the input file is switched to SYSUT3. All partially-edited records will be processed as they are read, inserting appropriate dictionary pointers (using the pointer and operand list records) to form an entirely edited record. If processing is within a macro, the edited record will be written on SYSUT3, otherwise on SYSUT1. During the processing of macros, a Macro Dictionary is built. The dictionary is closed and written on SYSUT3 when the MEND statement is encountered and overlay E4S is called to subset the dictionary.

During the processing of open code, a Permanent (Resident) Dictionary is built. When the end of text is encountered, overlay E4S is called to subset the Permanent Dictionary.

Overlay E4S reads dictionary segments from SYSUT3 and subsets this dictionary to contain only information that will be required for generation and conditional assembly. This subset dictionary is written on SYSUT3 and control returns to either E4M or E5P depending on whether the subset dictionary just written on SYSUT3 was the General Dictionary or the Permanent Dictionary, respectively.

Functions

The functions of this phase are designed

to perform the following:

- Establish the dictionaries necessary to complete the editing of the text.
- Subset the dictionaries for use in Phase E5P/E5/E5A/E5E.

The following dictionaries are established:

1. One global dictionary to contain macro mnemonics and global variable symbols.
2. One local dictionary for each macro to contain local variable symbols and sequence symbols, defined in the macro definition.
3. One local dictionary for the main portion of the program to contain relevant ordinary symbols in addition to local variable symbols and sequence symbols.

Both local dictionaries are never in core at the same time. See Appendix D for descriptions and illustrations of the format of the above dictionaries.

All symbols requiring dictionary action were collected in operand list dictionary collection records by Phase 3. For each macro definition, symbols are obtained, with the help of dictionary collection pointer records, from the associated operand list records. Dictionary definition entries are made in the global dictionary for the macro mnemonic and, at the point of definition, for all global set symbols. Dictionary definition entries are also made in the local dictionary for all macro prototype statement (symbolic parameter) operands, sequence symbols, and, at the point of definition, for all local set symbols. From its knowledge of the entries in the current dictionaries, Phase E4M directly calculates the structure of the associated evaluation dictionaries to be built in Phase 5. Using this information, E4M enters into its dictionary definition records an "a" pointer which points to the location within the comparable Phase 5 dictionary which will contain the appropriate evaluation information for the term.

Each macro prototype operand was assigned by Phase 3 a position number which reflects the order in which the operand appears within the prototype statement. This number is placed in a symbolic parameter definition entry instead of the "a" pointer. In Phase 5, this position number will be used to locate the required macro instruction operand that will replace the symbolic parameter wherever it is used. When a sequence symbol appears in the name field of a statement, it is entered in the local dictionary, its edited text is outputted, and the NOTEd location of this text is placed in the local dictionary entry for this sequence symbol in the space reserved.

During the dictionary processing, semantic errors (e.g., duplicate definitions of variable symbols) are detected and appropriate error action taken.

After processing, pointer and operand list records are dropped from Phase 4 output. The edited text of comments not for generation (e.g., .*) are not written on SYSUT3 and thus will not be generated.

These definition entries occur before the terms are referenced in macro instructions, model statements, or conditional assembly SETx, AIF, AGO, or ANOP statements. When a term defined in the dictionary is referenced, overlay E4M looks up the symbol in the dictionaries. When located in the proper dictionary, the "a" pointer or symbolic parameter location number, whichever is appropriate, is placed, with the help of the associated pointer record, in the partially-edited text record in the field reserved by Phase E3/E3A for such an entry.

The procedure for entering or locating an entry in a macro dictionary is described in Appendix D. Dictionary segments may overflow onto SYSUT3. The dictionary entries are backwards chained, so that recent entries are more likely to be in main storage. It may be necessary, however, to write the current segment on SYSUT3 and re-cycle previously written dictionary segments from SYSUT3 back into main storage. This re-cycling of dictionary segments is less frequent in large storage environments where more main storage can be made available for the dictionaries. When the MEND instruction trailer record is sensed, the local macro dictionary is closed out and control is transferred to overlay E4S which subsets the local dictionary segments. The NOTEd position of the macro definition fully-edited text is obtained from the macro mnemonic entry in the global dictionary and placed along with the current ACTR value in a header record attached to the first segment of the subsetted dictionary. The subsetted dictionary is then written on SYSUT3, its location NOTEd and entered in the macro mnemonic entry in the global dictionary.

When the open code flag record is read, the Relevant Ordinary Symbol Table segments are read in and the symbols hashed and entered in the (open code) local dictionary. Dictionary definition entries are made as before. When dictionary entries are referenced, "a" pointers are placed as before in the appropriate reserved fields in the partially-edited text that are pointed to by the associated dictionary collection pointer records. Whenever an ordinary symbol in the name field of a statement is encountered, it is looked up in the local dictionary. If an entry for this symbol is not found, this indicates that it is not a "relevant" ordinary symbol. If an entry is present, and represents the name of a DC/DS

statement, the attributes of the DC/DS name are already included in the entry. If attributes are not present in an entry, overlay E4M calculates the attributes and makes a second entry which contains the symbol attributes in the dictionary for this symbol.

The operand list records are processed as before for references to dictionary entries. The acquired "a" pointer, as before is placed in the proper field of the partially-edited text statement in the position pointed to by the appropriate operator list entry within the associated dictionary collection record. The now fully-edited text is outputted on SYSUT1 along with the interspersed source and error records. When the END assembler instruction is encountered, the local dictionary is closed out, as before, control is transferred to overlay E4S and the open code local dictionary is subsetted. The subsetted dictionary is outputted, as before. The location of the subsetted dictionary is NOTEd and the information is saved in COMMON.

The global dictionary is subsetted at the end of the phase, outputted on SYSUT3, and control is transferred directly to Phase E5P.

If overlay E4S is unable to completely subset the dictionary segments, output by overlay E4M, into the allotted areas, it takes the following action:

1. If a subsetted local macro dictionary overflows, it writes an error record on SYSUT3, NOTEs the position of this error record, and places this NOTEd value in the global dictionary macro mnemonic entry instead of the location of the partially subsetted dictionary. Control is returned to overlay E4M.

2. If the subsetted global or open code local dictionary overflows, overlay E4S writes an error record on SYSUT1 and copies source and error records, but not partially-edited text, onto SYSUT1 until the end of the file is reached. It then places an abort program code in general register 3, XCTLs to phase E4M which will XCTL to Phase E5P.

The formats of all records processed in this phase are described in Appendix D.

Routine Descriptions

ASGN2

Initialize origins for I/O buffers, directories, and hash tables.

CLRHT

Sets phase entrance code to first entry.

DCLOSE

General Dictionary closeout routine.

DCL3

Writes last dictionary segment.

DCL4

Writes next-to-last dictionary segment.

DCL6

End of text routine.

DCL8

Sets global declarations inactive and resets transient area pointers.

E4M

Phase closing routine.

E4SGO

Determines whether global or local entry.

FREECR

Frees the main storage area used for subsetting.

GETCOR

Gets main storage area for subsetting.

KASYB

Writes error record and abort flag.

MOCSUB

End of open code test.

MOC2

Sets end of open code flag.

MV4

Sets new common area base register.

NXTGRP

Tests for end of macro or open code.

PIVOT

Sets temporary common area base register to value of SYSREG.

RTRFIL

Reads the first dictionary statement from SYSUT3.

RZE17

Checks for operand list record.

RZE19

Writes edited test record.

RZE26

Processes operands for dictionary entries.

R0A

Clears out open code buffers.

R0D

Writes out error records.

R05

Processes macro instruction record.

R16

Processes global and local declarations.

SBSET1

Subset dictionary routine.

SSPGD

Subsets the Permanent Dictionary.

SSRTN

Tests if within open code.

SSTGD

Subsets the transient General Dictionary.

TSBERR

Stores the dictionary size in the dictionary header.

TSBSET

Subsets local dictionaries.

TSBST1

Reads all but the first segment.

TSBST2

Calls the subsetting routine.

TSBST3

Writes subsequent blocks of dictionary on SYSUT3.

TSBST4

Tests for open code dictionary.

TSBST5

Saves next NOTE position of SYSUT3.

TSBST7

Saves open code NOTE/POINT address.

WSSFIL

Writes subset dictionaries on SYSUT3.

PHASE E5P/E5/E5A/E5E CONDITIONAL ASSEMBLY AND MACRO GENERATION - CHARTS 10, 11, 12, 13, 14

Phase E5P requests a variable amount of main storage up to a maximum for dictionaries and I/O buffers. If an abnormal termination of assembly is requested, E5P calls E5E. E5P then reads the mainline global and local dictionaries from SYSUT3 and initializes them for E5. Control is then passed to E5.

Phase E5 reads text from SYSUT1. This text includes source, error and edited records. Source records are immediately written onto SYSUT2. When a macro instruction is encountered, control is transferred to overlay E5A. All other statements are processed for substitution, generation and conditional assembly by E5. Assembler edited text records are produced and written on SYSUT2. When the end of text is reached, control is transferred to Phase 07 via MAC and RTA.

Overlay E5A reads the macro dictionary from SYSUT3 and then the edited prototype statement for SYSUT3. A parameter table is built using the macro instruction and edited prototype statement. The input text file for E5 is changed to SYSUT3 for edited

text of the macro-instruction. Control is returned to E5.

Overlay E5E is called when an abort condition arises in E5. Text is read from SYSUT1. All source and error records are written on SYSUT2. All other records are bypassed. When the end of text is encountered, its source and associated error records are written on SYSUT2 and control is passed to Phase 07 via MAC and RTA.

Functions

The records that are read from SYSUT1 for this phase are source, error, and fully-edited text. Subsetted local and global dictionaries and macro definition edited text and associated error records, if any, are read from SYSUT3. The records that are written on SYSUT2 from this phase are source, error, and assembler edited text.

The functions of this phase are designed to:

- Initialize the phase.
- Take appropriate action if abnormal termination of the Assembly is necessary.
- Evaluate conditional assembly expressions.
- Perform conditional assembly.
- Generate a parameter table from a macro instruction - macro prototype pair of statements (refer to Appendix N).
- Generate assembler statements using macro definition edited text and the information in the global and associated local dictionary.

Overlay E5 reads the open code local dictionary into main storage immediately following the global dictionary and updates an available storage pointer to point to the next unused location in the block of main storage obtained by Phase E5P. Overlay E5 then evaluates conditional assembly expressions (Charts AA, AB, AC). If the expression was in a SETx statement, the "a" pointer associated with the set variable symbol is used to place the current set symbol value in its dictionary definition entry. When an AGO or AIF instruction is encountered and the evaluation, (if one is necessary), indicates that a branch must be taken, the NOTEd position of the fully-edited text named by the sequence symbol is obtained from the appropriate local dictionary. The text on SYSUT1 or SYSUT3 is re-positioned and the appropriate text is read in and processed.

When a macro instruction is encountered, overlay E5 NOTEs its exact location on the input file and whether the input file is SYSUT1 (for outer macro-instructions) or SYSUT3 (for inner macro instructions). Overlay E5 then makes a complete pass over the macro instruction text. Source and error records associated with the macro instruction are written on SYSUT2. When the End-of-Macro Instruction record is encountered, overlay E5 NOTEs the position where the reading of text was discontinued so that the input of text can later be resumed at the correct position. The appropriate utility data set is then re-positioned again at the beginning of the macro instruction text and control is transferred to overlay 5A.

Overlay 5A uses the "a" pointer associated with the macro instruction to inspect the associated macro mnemonic entry in the global dictionary. This entry contains a field which designates the location of the associated subsetted local macro dictionary on SYSUT3. If the entry is zero, this indicates that this macro mnemonic represents an undefined operation to the macro generator. If the entry field is not zero, overlay E5A reads in the first record of the subsetted local dictionary. This contains a dictionary header record which indicates the size of this local dictionary. If there is room available in the block of main storage acquired by Phase E5P, the complete local dictionary is brought into main storage and the available storage pointer is updated by the length of this local dictionary. The parameter table will be constructed at this location. This table (Appendix M) indicates the values to be substituted for macro prototype symbolic parameters when they are referenced in model statements or inner macro instructions. Entries are made for the first two parameters (& SYSNDX and & SYSECT). Then the NOTEd location of the pertinent macro definition prototype statement edited text is obtained from the local dictionary header record, the prototype edited text is read in and the parameter table is completed. For positional parameters, the values of the inner macro instruction operands are obtained from the appropriate dictionary, and for outer macro instruction operands they are obtained from the operand itself. Entries are sequentially made in the parameter table. As each prototype statement keyword is encountered, it is compared against each macro instruction keyword operand until a match is found. The values are then entered in the parameter

table. The cycles of comparisons to find the matching macro instruction operand corresponding to the next sequential prototype keyword begin where the last cycle left off; the operands being compared in a "wraparound" fashion. Because the entries in the parameter table are variable length entries, in position number (parameter) order, a separate length table with a two-byte entry for each parameter table entry is maintained. Each two-byte entry contains the length of its associated parameter table entry and thus can be used to locate its associated parameter table entry.

After the parameter table is completed, overlay E5A reads in the rest of the macro definition fully-edited text. Conditional assembly evaluation is performed as required, substitutions are made for references to symbolic parameters and system variable symbols, and the macro definition is expanded, producing assembler-edited text records for input to the assembler phases. If an inner macro instruction is encountered, the length table is placed behind the parameter table, and the entire cycle is repeated. Nesting of macro instructions can occur to any depth, provided there is sufficient room left in the block of main storage obtained by Phase E5P to enter the local dictionary associated with the inner macro instruction. If there is not sufficient room left, this information is noted for diagnostic purposes, the concerned local dictionary is not brought into main storage, further (deeper) nesting of macro instructions is discontinued, the input data set (SYSUT3) is NOTEd. Discontinued text of the next outer level macro is continued from where it left off.

When the ACTR value is exceeded within a macro expansion, the information is noted for diagnostic purposes, control is returned to the outermost macro instruction expansion, and processing continues. If the ACTR value is exceeded in open code processing, the information is noted for diagnostic purposes, an END assembler instruction record is created and inserted in the text stream on SYSUT2, and input is ended.

Whenever the processing of a macro definition is completed, generation of output text is resumed for the next higher level macro instruction or, if the outermost macro definition expansion has been completed, processing of open code fully-edited text is continued. After completely processing the fully-edited text input from SYSUT1, control is transferred to the first assembler Phase 07/07A/07B via MAC and RTA.

If abnormal termination is requested by Phase 4, Phase E5P calls overlay 5E instead of overlay E5. In this case, all text on SYSUT1 will be read, and only source records with accompanying error records will be written on SYSUT2 for input to the assembler phases.

Routine Descriptions - Charts 10, 11, 12, 13, 14

AGOST

Sets up to read at sequence symbol.

AIFST

Process AIF, and AGO statements. Branches to GOVAL to evaluate AIF expression.

BEGMAC

Sets up for building the Parameter Table. Reads in the Macro Dictionary (all segments) and initializes the Parameter Table with SYSNDX, SYSECT, and stores the macro instruction.

CALL

Return to overlay E5.

CSECT

Processes CSECT, DSECT, START, and COMMON statements. Branches to evaluate the name field if necessary.

ENDMI

Processes the end of the macro instruction record. It is used only on the preliminary pass to write source record and note the discontinued text.

ENDST

Branches to write the buffers on SYSUT2, services the utility files, and releases the main storage obtained for the General Dictionary, buffer areas, and COMMON area.

E5E

Passes control to overlay E5E.

MENDST

Processes MEND and MEXIT statements.

MINSTR

Performs a preliminary pass through macro-instructions to output source.

PHASE6

Initializes for E5 by restoring registers and checks for any errors. If entry is made from a subphase, and errors occurred in the subphase, appropriate action is taken.

PROTO

Reads in a prototype statement and NOTEs the prototype segment for the next prototype read. Checks for erroneous prototype.

PROTO1

Scans the prototype and macro instruction simultaneously determining the positional parameters to be entered into the table, and selects keywords to be entered in the table.

SETST

Processes SET statement. Branches to GOVAL to evaluate subscripted name, and to evaluate the operand.

SOURCE

Processes compressed source and error statements.

START6

Initializes for the phase by branching to read the input record, storing the beginning address of the record, getting the statement type and storing the statement flag. A computed GO TO is developed by using a table of DC statements labeled STTYPE and the statement type from the input record.

STTYPE

Computes GO TO for statement type.

USEBAS

Sets up main storage for E5 I/O buffers, COMMON area, and dictionaries.

Routine Descriptions - Charts AA, AB, AC

ADD

Performs addition.

ADVINP

Advances the input pointer.

ADVOP

Advances the operation pointer.

ARITOP

Processes arithmetic operators.

ATTPAR

Obtains attribute of a parameter.

BEGSUB

Processes substring expressions using the VALUAT routine for evaluation.

CHARST

Stores a character string.

CHFORC

Checks operator for parenthesis and the end of the expression.

CONTIN

Stores the product.

CRE

Updates pointer for intermediate results.

CSD

Processes a decimal integer or a character self-defining value.

DECINT

Processes decimal integers and character self-defining value.

DECINT

Processes decimal integers and character self-defining values.

DIV

Divide routine.

DOOPR

Performs the function of the operator forced in the table.

DOOPR1

Tests for arithmetic relational or logical operator.

FORCE

Tests if the operator forces the last operator in the stack.

GETVAL

Gets the value from the Parameter Table.

IATTBT

Makes the integer attribute available to the program.

INCPTR

Advances the pointer list and the GTLIST pointer for the next entry, sets GT to zero, and stores the pointer to GT in PTRLST.

LATTBT

Makes the length attribute available to the program.

META

Processes a SETA variable.

METAl

Converts to decimal.

META2

Advances pointers during SETA processing.

META3

Tests for character relational expression in SETC.

METB

Processes a SETB variable.

METB4

Tests for character relational expression in SETB.

METC

Processes a SETC variable.

METC4

Gets length of SETC variable symbol.

METINT

Initialize for dimensioned and undimensioned SET symbols.

MULTY

Multiplication routine.

NATTBT

Processes an N attribute.

NOTOPR

Creates computed GO TO for the appropriate value routine using a table labeled EVALBR.

PACK3

Stores addition result in pointer list.

PARMTR

Process parameter.

RELAT

Process relational operators EQ,NE,GT, LT,GE,LE.

SATTBT

Gets scale attribute.

SBEND

Processes the closing parenthesis of an expression and a comma between expressions.

SETA

Processes a SETA statement.

SETARE

Processes a substring left parenthesis and an A.R.E. flag.

SETB

Processes a SETB statement.

SETC

Processes a SETC statement.

SUBSC

Processes subscripted SET variables.

SUBTR

Subtraction routine.

SYMBL

Checks for parenthesis and end of expression.

SYSLST

Processes SYSLST.

TATTBT

Makes the type attribute available to the program.

TSTOP1

Determines if this operator has higher priority than operators in the operator list.

The Assembly portion (Phases 07 through DI) receives edited text from the generator, determines the amount of main storage to be reserved for each statement, assigns locations, evaluates expressions and constants, and completes the symbolic assembly process. Seven or more passes are required, additional passes being required when the Symbol Table and/or Symbol List Table overflow.

PROGRAM SEGMENT RTA

RTA contains the I/O routines used by the assembly phases for the system utility units. These routines perform the following functions READ, WRITE, SEEK, POINT TO READ, POINT TO WRITE, CHECK, and NOTE.

The routine labeled RTSEGA performs the initialization for this phase and contains the link (LINK macro) to Phase 07.

When loaded, RTA overlays MAC and remains resident in core during the assembly phases.

PHASE 07/07A07B - LITERAL COLLECTION
CHARTS 15 and 16

Phase 07/07A/07B adds work buckets, as required, to the input records, converts all self-defining numbers to their binary values, translates all substituted mnemonic operation codes created during the macro generation phase, builds Symbol List Table segments for use by Phase 8 and collects and inserts all literals into the program stream at the appropriate positions.

The input for this phase is from SYSUT2.

Text statements with their appended work buckets are written on SYSUT1. Symbol Table List segments are written on SYSUT3 along with any Literal Table overflow. Any op-codes generated in Phase E5 are looked up and replaced with the corresponding internal translation.

On completion of this phase, control is transferred to Phase 07I.

Functions

The functions of this phase are designed to:

- Add work buckets to the input records so that the size of the records will not be changed when the statement is evaluated in later phases.
- Convert any mnemonic operation codes that have not been processed, i.e., those created during macro generation or conditional assembly, to their hexadecimal equivalents.
- Collect all literals in a table according to four categories of total object length.
- Scan the appropriate category of the literal table for a previous definition of the same literal.
- When a LTORG or END assembler instruction is encountered, convert all previously collected literals to DC format and insert them into the text stream on SYSUT1.
- Convert all self-defining terms to their binary equivalents during a general scan, and insert them in the text statement, which is written on SYSUT1.
- Scan the operand field of the text statements for symbols requiring previous definition and enter those found in the Symbol List Table segment.
- Create dummy CSECT, ORG, LTORG text records when the END assembly record is read, to force the proper assignment of any literals to the first control section.

This phase attaches at least one work bucket to each edited record. Records may be broken before or between buckets. However, each work bucket occupies contiguous storage. The work buckets (Appendix E) which may be attached to the records are:

1. Statement (type A)
2. Operator/Delimiter (type B)
3. Length Symbol (type C)
4. Self-defining term (type D)
5. Ordinary Symbol (type E)
6. Literal (type F)
7. DC/DS Operand (type G)

Every edited statement has at least a type A work bucket attached. Each machine operation has as many types B through F buckets attached as required to duplicate the operand field. Each DC/DS statement has one type G bucket per operand. Each type G bucket is followed by types B through E as needed for duplication factors, length, scale or exponent modifiers, and/or address constant expressions.

To create the Literal Table (Appendix I), this phase scans the operand field of the statements for literals, collects each literal and enters it into the Literal Table segment according to the total object length

of the literal (including duplication and multiple constants). This segment is divided into four strings. These strings represent four categories of total object lengths (divisible by 8, and for odd multiples of 4, 2 and 1 byte). When a segment is full, it is written on the overflow file SYSUT3.

Before a literal is entered into the segment (Figure 5), the appropriate string of the segment is scanned for a previous definition of the same literal. When there is more than one Literal Table segment, the current segment in main storage is scanned, written out on the overflow file SYSUT3, and the other segments are read in one at a time and scanned until a duplicate is found or until all segments have been scanned. The current segment is finally brought back into main storage to be completed. If a duplicate is found the literal is not collected.

In every case, a pointer representing the string in which the literal (or its duplicate) was entered and its displacement within that string (i.e., the sum of the total object lengths of all previous entries in that string) is appended to the text statement. This locater information is used in a later phase to determine the address of the literal.

All previously collected literals are converted to DC format (Appendix D) when a LTORG or END assembler instruction is encountered. The literals are extracted from the Literal Table segments by strings in the order of the four categories (8-4-2-1) and are inserted into the text stream which is on SYSUT1.

Figure 5 is an example of the steps involved in building the Literal Table segments.

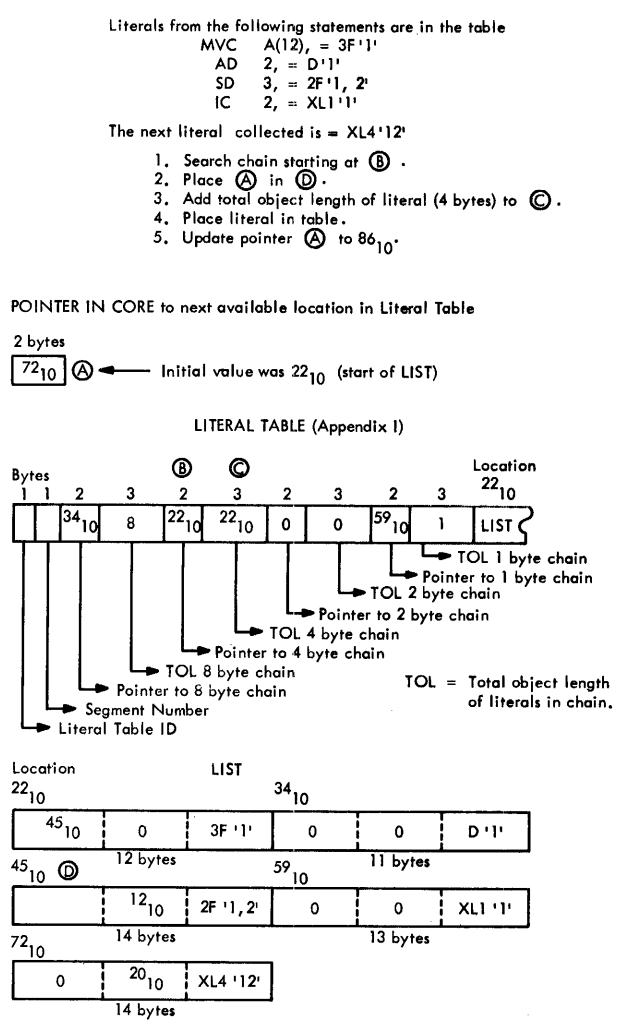


Figure 5. Building the Literal Table

Routine Descriptions

FINTST

Checks for final literals, then writes them out with a dummy CSECT, ORG, or LTORG.

GETEXT

This subroutine is common to all assembly phases of this program. It is called to retrieve the next logical record from the input buffer and store it in the output buffer.

LITORG

Process the literals at the END assembly indication.

MAIN12

Start of overlay 07A.

MAIN15

Branch out according to type of instruction.

MARKCF

Marks the location for adjustment of the moved record.

OBM - Output Buffer Management

This subroutine is common to all input/output routines in the assembly phases of this program. When OBM is entered, the RLI is in register GRO and upon exit, register GR2 contains OCT+4 words (Output Control Table). The return register is GRX.

OPSCAN

Scans the Operand field for literals.

PHCD

Switches read and write data sets.

PHCG

Exchanges read and write buffers.

PHCLS

This subroutine closes Phase 07/07A/07B. It includes logic to determine the amount of main storage to free and which phase to call.

PHC2

Determines if this is the first Symbol List segment.

PHSIN

This is the initialization routine for this phase. It includes routines to set pointers for the Symbol List segment in I/O Parameter Table and Resident Control Table (RCT) input buffers and buffer management, and first output buffer and buffer management. It also determines the length of the Literal Table and sets table pointers. Sets the utility I/O unit DCBs in RCT and R/W DECBs.

PTWBCF

Adds work buckets to the record.

PUTREC

This subroutine is common to all of the assembly phases of this program. It is called to retrieve a logical error record and store it in the text output buffer. For this phase it is also used to output the three dummy records (CSECT, ORG, LTORG) created when the END assembly record is encountered.

RELREC

This subroutine is common to all of the assembly phases of this program and is called to release a record from the input buffer.

SDVCF

Converts self-defining values or scans the non-address type constants.

SLNAME

Puts name in the Symbol List.

SYLIST

Enters the symbol in the Symbol List Table segment.

TXTCF

This is the main routine for processing the statements in this phase. The routines included are:

LITORG

OPSCAN

PTWBCF

SDVCF

SYLIST

WRTEXT

Writes text and error records.

PHASE 07I - BUILD SYMBOL LIST TABLE - CHART 17

Phase 07I is called Interlude since it is not an overlay and it does not pass text. Interlude builds the Symbol List Table in sections from Symbol List Table segments which were output from Phase 07/07A/07B. The Symbol List Table is used by Phase 08.

Input for this phase comes from SYSUT3 and consists of Symbol Table segments. The Symbol List Table sections are written on SYSUT3.

The volume of Symbol List Table segments may cause Interlude to build a number of Symbol List Table sections. Duplicate occurrences of symbols within one Symbol List Table section are eliminated.

On completion of this phase, control

is transferred to Phase 08/08A/08B.

Functions

- Read the Symbol List Table segments (1,000 bytes) from overflow file SYSUT1.
- Build Symbol List Table sections (3,000 bytes).
- Write filled Symbol List Table sections back on SYSUT3.

Routine Descriptions

PHCLS

Phase closing routine which includes transferring control to Phase 08.

PHSIN

Initialization routine.

PLUGCF

Clears the main storage area reserved for building the Symbol List Table section.

RDSLS

Reads the Symbol List Table segments from SYSUT3.

SYL10

Links to the subroutine to read the Symbol List Table segment.

SYL13

Gets the symbol count and sets the symbol pointer.

SYL15

Hashes the symbol and gets the hash entry contents.

SYL20

Compares the entry in the chain with the hashed symbol. Duplicates are not entered in the chain again.

SYL25

Tests for the end of a chain or the absence of a chain. Enters the symbol in the next available storage position and sets the chain pointer and hash entry. Increments the section symbol count. If the section is full, it is written and the area is cleared.

SYL30

Reduces the segment symbol count after a symbol has been entered in the chain or its duplicate found.

SYL35

Increments the total symbol count. If a partially processed section remains, the routine increments the section symbol count and moves it to the section, writes the section on SYSUT3, and branches to call Phase 08.

WTSLT

Writes a filled Symbol List Table section on SYSUT3.

PHASE 08/08A/08B - STORAGE ASSIGNMENT CHART 18

Phase 08/08A/08B evaluates all expressions requiring previous definition of symbols, assigns storage addresses to all instructions, constants, and storage areas, and creates External Symbol Dictionary (ESD) segment(s).

The input for this phase for the first, or only, pass is SYSUT2. Text is written on SYSUT1. Symbol List Table sections and External Symbol Dictionary segments are written on SYSUT3. SYSUT2 and SYSUT1 alternate functions on succeeding passes, if any.

On completion of this phase, control is transferred to RTB.

Functions

The functions of this phase are designed to perform the following:

- Assign relative storage addresses to all instructions, constants, and areas.
- Evaluate all expressions requiring previous definition of symbols.
- Make entries to the External Symbol Dictionary for symbols in the operand fields of ENTRY and EXTRN statements, V-type address constants, each named control section and dummy control section, blank common, and for an unnamed control sec-

tion and dummy section, if present.

The following factors enter into the storage address assignment function:

- Control section.
- Statement object length and programmed location counter adjustment (ORG, CNOP).
- Boundary alignment.

For the first appearance of each control section, Phase 08 establishes a location counter starting at zero, and records the control section name, number and current length in an ESD table segment (see Appendix I). For a named CSECT, START or DSECT, the Symbol List Table is scanned to determine whether a new section has been requested or a previous one resumed. If it is a resumption, the location (within the ESD table), of the previously established ESD record for this control section, is calculated and the current value of the location counter for this control section is obtained. The location of an ESD record for Private Code, blank common, or unnamed DSECT is calculated from information held in core storage. ESD processing is on a Demand-overlay basis (overlay E8B), with ESD segments written on SYSUT3 overflow data set. If a needed ESD segment is not in main storage, the current segment is rolled-out (written) on SYSUT3 and the previous segment brought in.

The object length of machine instruction statements is determined from the actual operation code. The object length of declarative statements (e.g., DC) depends on the constant type, length, and, if present, duplication factor and length modifier expressions. Symbols used in these expressions, in scaling and exponent modifier expressions, or in the operand fields of ORG, CNOP, START, or EQU statements must have been previously defined. These expressions are evaluated by overlay 08A and the evaluations placed in the proper work bucket. Expressions in the operand field of ORG statements are evaluated and the current location counter field in the proper ESD record is updated.

Boundary alignment of statements is done on the basis of statement (or constant) type, or, in the case of CNOP, based on the evaluation of the operand field. Information about bytes skipped for alignment is carried in the proper work bucket.

In evaluating expressions, searching the Symbol List Table is said to be done in one of three modes:

- Normal mode
- Iterate mode
- Re-iterate mode

The conditions associated with each are as follows:

Normal Mode: The statement name entry, if any, is looked up in the current Symbol List Table section. If the name is found, its attributes are attached to the table entry. All name entries are checked for validity on the first pass over the text. When an expression contains a symbol requiring previous definition, the current Symbol List Table section is searched and the associated length and value attributes are placed in the proper work bucket. When the value of all Symbols in each expression of a statement have been found, the expression(s) are evaluated, the length of the statement is determined, and the length and value attributes are attached to the work buckets. The location counter for the current segment is advanced by the total length of the statement.

Iterate Mode: If a symbol in the operand field of a statement is not in the Symbol List Table, the Symbol List Indicator (SLI) for the statement being processed is set. From this point, the remainder of the text is processed, values are substituted in the proper work bucket for symbols found in the Symbol List Table section, but expressions are not evaluated.

Re-iterate Mode: Input is initially from SYSUT2 and the entire data set is passed as output to SYSUT1. When a logical end-of-data is encountered, the text data sets are re-positioned at the beginning of text and SYSUT2 and SYSUT1 switch functions (i.e., SYSUT 1 becomes input and SYSUT2 becomes output). The next Symbol List Table section is read and searched for the name that could not be found in the previous table section(s). If the name is found, its attributes are attached to the table entry. Evaluation, length determination, location counter advancement, and table updating continue until another symbol cannot be found, at which time the above process is repeated. Thus the entire text is passed for each Symbol List Table section searched.

If, for any reason, an expression cannot be evaluated, an entry is made in an error record associated with the statement. The current location counter field in the ESD entry is not updated for that statement.

When entering symbols from the operand field of ENTRY, EXTRN, and V-type constant statements into the ESD table, ENTRY symbols are entered each time they are encountered; no check is made at this time for duplication. However, only one entry in the ESD Table is made for EXTRN and V-type constant statements. Should the same symbol appear in

both an EXTRN and a V-type constant statement, the ESD Table entry is an EXTRN. If a DSECT is encountered with the same name as a previously encountered CSECT, it is entered as an unnamed (illegal) DSECT. If a CSECT is encountered with the same name as a previously encountered DSECT, it is treated as a private code (i.e. unnamed) control section.

A maximum of 255 entries may be made in the ESD Table (excluding symbols appearing in the operand field of ENTRY statements). Entries in excess of the 255 maximum (except for ENTRY) are flagged as errors and not entered in the table.

Routine Descriptions

MAIN10

Reads the next record and initializes the statement.

MAIN12

Exit from Phase 08.

MAIN15

Reads the next Symbol List segment.

MAIN20

Determines the process type.

Tests the record read by routine MAIN10 to determine whether it is to be bypassed or processed. If it is to be processed, it is further tested to determine whether it contains an assembler operation code or a machine instruction. If the statement contains an assembler operation code, exit to MAIN50. If the statement contains a machine instruction and Normal Mode is set, the operation code is checked to determine its format (RR, RX, SI, SS).

MAIN40

Advances the location counter. If the statement contains a name, a length attribute is formed and placed in the statement work bucket.

MAIN45

Exits to NAMECF to look up statement name in the Symbol List Table and, if the table entry is incomplete, inserts the missing information (length attribute, ESD-ID, value attribute, adjective code).

MAIN50

Enter from MAIN20 if the statement contains an assembler operation code, if this operation is Hexadecimal 16 or greater, the actual operation code is determined and a branch occurs to the appropriate subroutine. Assembler operation codes below Hexadecimal 16 are bypassed.

PHCLS

Phase closing routine. Closes text data set, frees main storage, and transfers control to RTB.

PHSIN

Entry point to Phase 08. Performs initialization.

PROGRAM SEGMENT RTB

RTB contains the DCBs and DECBs for the system output units SYSPRINT and SYSPUNCH. The routine labeled RTSEGB performs the initialization for this phase and contains the link (LINK macro) to Phase 09.

RTB, once loaded following Phase 08, remains resident through the remainder of the assembly phases. The routines using the DCBs and DECBs contained in RTB are incorporated in those phases performing output functions.

PHASE 09 - BUILD SYMBOL TABLE - CHART 19

Phase 09 reads the text from SYSUT2 or SYSUT1. Each symbol from the name field of a statement is entered in a symbol table, together with its value and other attributes. A masking formula is applied to a symbol to obtain an entry point in a pointer table. This table contains a pointer to the symbol table, where all symbols are entered sequentially. Overflow of the symbol table is written on SYSUT3. PUNCH and REPRO statements preceding the first control section are processed by this phase. This phase punches symbol cards for later use by TESTRAN.

Functions

The input for this phase if from SYSUT2 or SYSUT1 depending on the number of passes made of the text statements during Phase 08. The format of these input records is described and illustrated in Appendix D. The functions

of this phase are designed to:

- OPEN SYSPRINT and/or SYSPUNCH if the LIST and/or DECK or LOAD options are selected.
- Process and write on SYSPUNCH the data from all PUNCH or REPRO assembler instructions appearing before the first control section.
- Build Symbol Table segment(s).
- If the number of Symbol Table segments is greater than one, all segments are written on the overflow file SYSUT3. Otherwise, it remains in main storage.
- Optionally writes the TESTRAN Symbol Table on SYSPUNCH (Appendix I).
- Builds a Literal Base Table in segments. The table contains locater information for Phase 09I to use in developing a table of assembled addresses for the beginning of each literal pool and its strings.
- Writes the Literal Base Table segments on SYSUT3 if the number is greater than one.

The Symbol Table consists of two parts: a random pointer table (Hash Table - Appendix J), and a sequentially forward-chained symbol table. Each name (a literal with an * reference to the location counter is now in DC format with the statement number as a name) is hashed into the Pointer Table and the appropriate pointer is examined. If it is zero, no other previous names have the same hash table location. The current name and its attributes are then placed in the Symbol Table segment at the next available location and the address of that Symbol Table location is placed in the Pointer Table. If the Pointer Table is not zero, the routine searches the Symbol Table chain in the current segment for a matching name. If none is found, the name is entered in the next available location in the current Symbol Table segment. The address of the new location is then placed in the pointer field of the previous entry in the chain. If a duplicate symbol is found, the name is ignored. Because this is a read-only phase (for text), the duplicate symbol must be diagnosed in a subsequent phase.

The format for the Literal Base Table segment is described and illustrated in Appendix I. The literal work bucket in a text statement (Appendix E) contains indicators which describe the location of the literal relative to its string (8-4-2-1) within the literal pool associated with the particular LTORG or END assembler instruction. The location field in the Literal Base Table gives the address of the 8-byte string (and by implication, the addresses of the 4-2-1 byte strings) relative to the beginning of the control section in which a particular pool is located.

This table is passed in core to Phase 09I if it consists of only one segment. Otherwise, all segments are written on the overflow file SYSUT3.

For each PUNCH or REPRO assembler instruction which appears before the first control section, one card appears in the object output before all other output (e.g., before ESD cards).

Routine Descriptions

COLENG

Creates computed GO TO for length attribute collection.

DUMP

This subroutine is used to dump a specified number of bytes when called.

DUNSYM

Processes LTORG statement by PUTLB, GETWKB and TESRAN.

GETPT

This subroutine contains the logic for handling the records in the input buffer. It establishes the pointer at the next logical record in the input buffer.

GETWKB

This subroutine is common to all assembly phases of this Assembler and is used to get the information from the various work buckets appended to text statements.

IEXTRN

Looks for a name symbol work bucket for possible symbol entries.

LASTAB

Executes the end of data processing including WTST, TESRAN, PUTLB and PHCLS (calls the next phase).

MACHOP

Sets DOPCH switch for machine type operation.

NOBIT

Tests for a conditional computed GO TO, a possible symbol to enter, and a title card to process (the first time only).

PHCLS

Phase close routine.

PHSIN

Phase initialization.

PUN1

Processes the PUNCH statements. Branches to SYSOUT to output the punch statements and resets the punch buffer.

PUTLB

This subroutine makes entries into the Literal Base Table segment. It branches to write the segment on the overflow file when the segment is filled and resets the buffer.

PUTSYM

Hashes symbol and looks for duplicate. Puts symbol in table unless already there.

REP1

Processes the REPRO statements.

SIZUP

Initialize hashing routine.

SOMSYM

Finds out if there is actually anything to enter in the Symbol Table and collects the information for the Symbol Table entry.

TESRAN

Functional routine for TESTRAN.

TEXTI

This is the label of the first instruction in the main logic of this phase. The first section gets the input record, checks for end of data, tests the record-type, counts the statement, tests the operation type and branches to the particular routine for the particular operation. The routines that make up the main logic of this phase are:

NOBIT	SOMSYN	PUTSYM	IEXTRN
DUNSYM	COLENG	TY1	LASTAB

MACHOP

TY1

Processes the first Title card.

WTST

This subroutine branches to write the Symbol Table segment on the overflow file when the current segment is filled or at the end of the phase.

PHASE 09I - ESD (EXTERNAL SYMBOL DICTIONARY) OUTPUT - CHARTS 20 AND 21

Phase 09I reads the ESD entries put on SYSUT3 by Phase 08. This phase prints out the ESD items and ESD cards. The ESD items include CSECTs, DSECTs, ENTRYs, and EXTRNs. Phase 09I adjusts the entries in the Symbol Table by the starting location of the control section that the Symbol Table items is located in. 09I also adjusts the Literal Base Table. Phase 09I does not pass the text stream or produce any diagnostic messages.

On completion of this phase, control is transferred to Phase 10.

Functions

The input for this phase is from SYSUT3 in the format described and illustrated in Appendix D. The functions of this phase are designed to:

- Read in each ESD Table segment which was created in Phase 08/08A/08B and build an ESD Adjustment Table in main storage.
- Read in each ESD Table segment again and process it against each Symbol Table segment which was created in Phase 09.
- Adjust each Symbol Table segment to reflect the starting assembled address for the control section to which it belongs.

- Write the adjusted Symbol Table segments on SYSUT3.
- Process, format and output the ESD segments on SYSPRINT and SYSPUNCH.
- Read in the Literal Base Table (it may have been passed to this phase in main storage) and create the Literal Adjustment Table which gives the assembled location of all literal pools.
- If the Literal Adjustment Table exceeds one segment, write all segments on the overflow file SYSUT1.

The entries in the ESD Adjustment Table consist of ESD/ID and assembled addresses for all control sections except COM and DSECTS. The first entry contains the value from the Operand field, of the START instruction. If this instruction was used, the value was passed to this phase from Phase 09. The HLF field (highest location for this control section) of the ESD Table entry for this first control section is then added to the START instruction value to create the starting assembled address for the second control section. The HLF field for the second control section is then added to this value to create the starting assembled address for the third, etc., until all ESD Table segments have been read.

As the first ESD Table segment is being processed against the Symbol Table segments, each symbol in the Symbol Table segments is adjusted to reflect the starting assembled address for the control section to which it belongs.

Processing the ESD Table segments involves hashing the symbol in each ENTRY record. Then if it is present in the Symbol Table segment, its address is obtained and placed in the ENTRY ESD record.

The Literal Adjustment Table is created by adding the starting assembled address for the control section in which the LTORG or END is encountered to the location field of the Literal Base Table to produce the starting assembled address of the 8-byte string associated with that LTORG or END statement. The lengths of the remaining strings are added to this address to obtain the other entries (for this literal pool) to the Literal Adjustment Table. Figure 6 illustrates the action schematically.

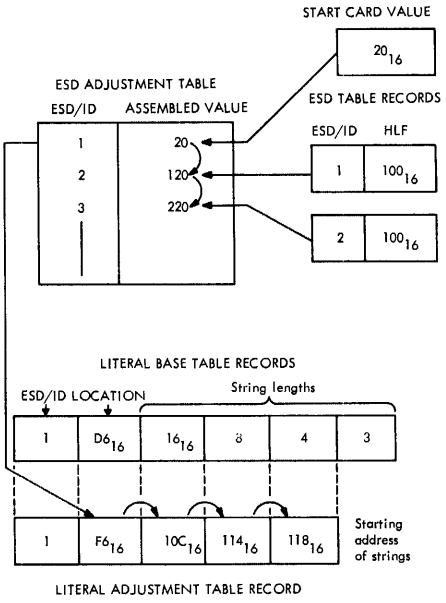


Figure 6. Table Creation During Phase 09I

Routine Descriptions

CALLF8

Writes the Symbol Table.

CALL10

The last instruction of the phase. A branch is taken to write the Symbol Table and to PHCLS to close the phase and call Phase 10.

EI2B

Sets ESD pointer.

EI3A

Builds Adjustment Table entry.

EI3B

Determines ESD type.

EI3CF

Processes LD type ESD entries.

EI4

Sets ESD Adjustment Table item (ID, START value).

EI8A

Gets next ESD segment.

EI11P

Gets the Literal Base Table.

FSTLP

Sets the ESD pointer and checks for more than 16 ESDs remaining.

GETLBT

This subroutine provides the Literal Base Table records in the LTB buffer. If the table exceeded one segment, all segments were written on SYSUT3 in Phase 09.

LOOPSR

Searches the ESD Adjustment Table with a Literal Base Table indicator.

NXESD

Determines if the number of ESD items is greater than 16.

NXTST

Determines if the item needs adjustment.

NXTST1

Tests to see if this is the last segment.

PHCLS

This subroutine closes Phase 09I by branching to write out the Literal Adjustment Table (if necessary), and calling the next phase.

PHSIN

This subroutine initializes for Phase 09I. It includes instructions to set pointers for the Literal Base Table, the Literal Adjustment Table and the ESD Adjustment Table.

PRNTHD

This subroutine prepares the heading records to be written on SYSPRINT.

PUTLAT

This subroutine writes the Literal Adjustment Table segments onto SYSUT3.

RDESD

This subroutine reads the External Symbol Dictionary Table segments from SYSUT3 into the input buffer.

RDST

This subroutine reads the Symbol Table segments from SYSUT3 into a buffer.

RELFND

Sets the Literal Adjustment Table.

SECPAS

The first label of the logic that processed the ESD Table segments against the Symbol Table segments. It includes the routines to adjust the Symbol Table entries (NXTST), process ESD type (E13B), and create the Literal Adjustment Table using the Literal Base Table (ET11P, LOOPSR, RELFND).

SETSEG

Sets the number of ESDs.

SETSGR

Branches to read the ESD segment, makes an entry into the ESD Adjustment Table (EI4), points to the next ESD item (EI8A).

SETTWO

Sets the number of items in the ESD segment into work bucket NOESD.

STADJ

This subroutine adjusts the symbol in the Symbol Table segment.

SUB

This subroutine prepares the card image to be written on SYSPCH.

SYSLST

This subroutine is common to Phases 09 through DI. It writes the records prepared by the phase to be printed.

WTST

This subroutine writes the adjusted Symbol Table segments onto SYSUT3.

PHASE 10 - SYMBOL SUBSTITUTION - CHARTS 22, 23, 24

Phase 10 reads text from either SYSUT2 or SYSUT1, and performs any required CSECT

adjustment of the location counter. Any referenced symbols are looked up in the Symbol Table and their values are placed in the text work buckets. Phase 10 also writes all cross-references on SYSUT3 (overflow file). Phase 10 passes text once for every Symbol Table segment.

On completion of this phase, control is transferred to Phase 10B.

Functions

The functions of the phase are designed to:

- Read text from SYSUT2 or SYSUT1 and the Symbol Table segments (adjusted in Phase 09I) and the Literal Adjustment Table from SYSUT3.
- Look up each operand field symbol in the Symbol Table segments and attach the value to the text statement. The Literal Adjustment Table is used to obtain the assembled storage address of literals which, up to this phase, represent only the displacement within a particular literal string. (See Literal work bucket of Appendix E.)
- Detect and flag symbols that have been defined more than once and symbols used more than once in the operand field of an ENTRY statement.
- Output cross reference information on the overflow file (SYSUT3) for sorting and listing by the Post Processor Phase. The input for this phase is from SYSUT2 or SYSUT1 depending on the number of passes made of the text statements in Phase 08. The format of these input records is described and illustrated in Appendix D.

The values are attached to symbols in instruction operands which do not require previous definition. Those that required previous definition were collected in Phase 07 and evaluated in Phase 08.

The text statements must be passed once for each Symbol Table segment created in Phase 09. On each pass, the previous input and output system utility files are switched. When the utility that was input on the first pass is used as output, it is known as iterate mode.

In performing the look up operation, this phase checks also for the presence of Names and Operands when required or their absence, when required.

Routine Descriptions

ABR

Processes the Statement Type work bucket (type A).

ACR

Processes the TITLE card records as they appear.

AC01

Process TITLE cards.

ADR

Processes the DC/DS type work bucket (type G).

AFR

Processes the literal type work bucket (type F).

AGR

Processes the Ordinary Symbol (Name) type work bucket (type E).

AG01

Looks up symbol in Symbol Table.

AG091

Checks for duplicate symbols.

AHR

Processes the Length Symbol type work bucket (type C).

AH01

Gets value from Symbol Table.

AH011

Sets pointers to statement fields.

AMR

Increments the location counter for *symbol.

AM01

Gets symbol value and adds length attribute.

CLSTXT

This subroutine closes the text statement data set after each pass. It includes logic to switch the files (SYSUT2 and SYSUT1) and interchange the I/O logic.

DPCHEK

This subroutine checks each symbol for multiple definitions. The multiple definition flag is MDON.

GESRCE

This subroutine makes available the next logical text statement record.

GETLAT

This subroutine makes available the next logical record in the Literal Adjustment Table.

GETWKB

This subroutine is common to all assembly phases of this assembler and is used to get the information from the various work buckets appended to text statements.

LOGERR

This subroutine is common to all assembly phases of this assembler and is used to put error flags in the generated text flow.

LOOKUP

This subroutine looks for a symbol in the Symbol Table segment that equals the symbol from the text statement. Included is logic to hash the symbol, to develop the address of the hash pointer and to check for synonyms.

ML00

This is the label of the first instruction in the main logic portion of this phase. These instructions branch to PHSIN to initialize this phase. The routines that follow are labeled ML01, 02, 31, 34, 04, 05, 10, ABR, ADR, AEC, AFR, AGR, AHR, ACR, AMR.

ML01

Tests the text statement for the presence of a machine operation. If not, and the first pass switch is on, it checks for the required presence or absence of a name and/or operand (ML010, ML011, ML012, ML013).

ML02

Tests for a TITLE, LTORG, PRINT, REPRO or LITR (any literal) card and branches to the appropriate routine to process the particular type.

ML04

Makes the cross-reference entries. Branches to look symbol up in the Symbol Table and checks for multiple definition (ML048).

ML05

Turns on name reference for cross-reference. Branches to get the next work bucket.

ML010

Tests for required Name field.

ML10

This routine either terminates the job or switches TEXTIN and TEXTOUT to make another pass over the text.

ML011

Sets pointers to statement fields.

ML012

Checks if an operand is required.

ML013

Tests if blank operand is required.

ML 20

Tests if this is a LITR operation.

ML24

Turns off literal switch, EXTRN flag, and first operand switch.

ML031

Branches to get the Literal Adjustment Table.

ML034

Steps statement counter.

ML048

Checks for multiple definitions.

PHCLS

This subroutine closes Phase 10 by writing out the cross-reference block and calling the next phase.

PHSIN

This subroutine initializes for Phase 10. It includes logic to set pointers in the various buffers and branches to read into main storage the Symbol Table segment and the Literal Adjustment Table.

PUTXRF

This subroutine writes the cross-reference information when required.

RDST

This subroutine is common to Phase 09I and Phase 10. It is used to read the Symbol Table segments into main storage.

SETAST

This subroutine sets the Symbol Table save area when the symbol in the operand field is an asterisk.

SETNAM

This subroutine is used to obtain the symbol name from the text statement.

STGET

This subroutine looks up the symbol in the Symbol Table. If the statement is an EXTRN, a cross reference is made. Included is logic to test for additional Symbol Table segments.

SYMSCN

This subroutine scans the symbol in the operand field. If an illegal character appears in a symbol, the scan terminates. It places the symbol in STSYM.

WTXRF

This subroutine writes out the last block of cross reference information at the end of the phase.

PHASE 10B - EXPRESSION AND DECIMAL CONSTANT EVALUATION - CHART 25

Phase 10B evaluates all expressions not requiring previous definition, with the exception of some expressions used in address constants. It processes the USING, DROP and listing control instructions. It also produces source records from the edited text records created by macro expansion and conditional assembly substitution. Finally, this phase rearranges or substitutes the format of some edited text records in preparation for processing by succeeding phases.

The input for this phase is from SYSUT2 or SYSUT1 depending on the number of alternating passes made of the text statement records in Phase 10. The output is written on SYSUT2 or SYSUT1, whichever was not the input data set.

On completion of this phase, control is transferred to Phase 21A.

Functions

The functions of the phase are designed to:

- Read the text records from SYSUT2 or SYSUT1 and locate the next logical record.
- Evaluate the expressions not requiring previous definition and place the value in an evaluation work bucket (Appendix E).
- Process the USING, DROP, PRINT, SPACE, EJECT, TITLE, PUNCH, MNOTE, EQU, ORG, DSECT, CSECT, and END assembler instruction statements.
- Create object records for the USING, DROP, CSECT, DSECT, COM, ORG, EQU, CNOP, DS, TITLE, PRINT, and END assembler instructions.
- Rearrange the format of the text records for the TITLE, PUNCH, and MNOTE assembler instructions.
- Update and output the Register Availability Table (Appendix I) for each USING or DROP assembler instruction.
- Update and output the Switches Table (Appendix I) for each PRINT, SPACE, TITLE, or EJECT assembler instruction.

The evaluation work bucket replaces the

type B,C,D, and E work buckets that were appended to the text records in Phase 07/07A/07B with the exception of

1. A type B work bucket added to record because of a comma, blank or parenthesis, used syntactically.
2. For SS type instructions, additional parenthesis (type B) work buckets may be inserted for syntactical purposes.
3. An ADCON that contains a reference to the location counter (*) and a duplication factor greater than one. All other DC/DS statements drop all appended work buckets from their output except the type A and G which already contain the evaluated value.

The evaluation work bucket contains the assembled address resulting from the expression evaluation and indicators. For ADCONs, the indicator tells the linkage editor what values to add and/or subtract from this value when relocating the object module. For other statements, the indicator is used by succeeding phases to determine the ESD identification number.

Each time a USING or DROP assembler instruction is processed, the Availability Table is updated to indicate which registers are currently available to the assembler and what values they are specified to contain. This table is written out in the text stream each time it is updated, to be used by succeeding phases to develop addresses in the base/displacement form.

The Switches Table is updated to indicate what listing actions are to be taken (e.g., space, eject) and which listing options are currently in effect (e.g., Gen. Data). This table is written out in the text stream each time it is updated, to be used by succeeding phases to control the assembly listing. A TITLE statement updates the table in the same manner as an EJECT statement. In TITLE, PUNCH, DC, DS and MNOTE processing, double quotation marks (") and double ampersands (&&) are replaced by ' and &, respectively.

The source records (Appendix D), created in Phase E2 or in this phase, are used to create the listing image records for the right-hand side of the print line of the assembly listing. An object record (Appendix D) is used to create the listing image record for the left-hand side of the print line. These object records replace the associated edited text record and its appended work buckets.

The format of the PUNCH statement is rearranged so that succeeding phases can treat PUNCH and REPRO statements similarly.

In processing an MNOTE statement, an error record is produced if the severity code is not (*).

In processing an EQU statement, the expression in its operand field is reevaluated. If the expression is absolute or in error, the result of the Phase 07 evaluation is retained. However, if the expression is relocatable, the new reevaluated value, reflecting the proper assembled address and ESD/ID, is retained.

When the END assembler statement is processed, the operand is examined for validity and the associated dummy CSECT and ORG records (created by Phase 07/07A/07B) are dropped. Only the edited text statements for the END and the associated LTORG records are retained for further processing.

Routine Descriptions

BLDIMG

This subroutine builds a source image from record types 100, 111, and 011. It changes edited record type 111 to generated record type 011 and generated record type 100 to generated record type 010. However, if the edited record is a literal, the generated record type is made 000. Register GRA points to the first byte of the input record.

CLEAR

This subroutine clears an 80 byte area.

ECCWX

Sets up to branch to appropriate evaluation routine.

EEND

Processes END statement.

EENQ

Processes END statement.

GETBKT

This subroutine is used to make available to the program the next work bucket from the input record. It contains a branch to the subroutine GETWKB which is common to all assembly phases of this program and contains the logic necessary to interrogate the work buckets that are appended to the text statements.

GETPTR

This subroutine sets the pointer at the next logical record in the input buffer.

It is used by ML1 in this phase.

GETXTM

This subroutine moves the text record into the output buffer. It includes the common subroutine MOVE that moves any size block within main storage from the location indicated by register GRY to the location indicated by register GRZ.

ML1

This is the label of the first instruction in the main logic portion of this phase. These instructions branch to get the next logical input record and set and reset switches and test for record type.

ML2

Processes record type 011.

ML2C

Processes record type 000.

ML2G

Processes record types 100, 101, 110 or 111.

ML2T

Constructs for printing before processing record types 110 or 111.

ML3

Processes the contents and tests for type of assembler operation.

ML3A

Branch to appropriate routine for type of assembler operation.

ML3F

Processes DC, DS, or LITR.

ML9

Checks for any error in the statement being processed.

OBM

This subroutine is common to all input/output routines in the assembly phases of this program. This routine is entered with the RLI in register GRX and exits with OCT+4 words (Output Control Table) in register GRZ. The return register is SRR. In this phase it is called by GETXTM, PUTERR, and PHCLS.

PHCLS

This subroutine closes Phase 10B and calls the next phase.

PHSIN

This subroutine initializes for Phase 10B. It includes instructions to set pointers in the input/output buffers.

PUTBKT

This subroutine is common to all assembly phases of this program and is used to append the various work buckets to the text statements.

PUTERR

This subroutine is common to all assembly phases of this program. It is used to retrieve a logical error record (or any record) and store it in the text output buffer.

PHASE 21A - MACHINE INSTRUCTION EVALUATION - CHART 26, 27

Phase 21A reformats storage addresses used in machine instructions to base displacement format, converts fixed and floating point constants to their binary equivalents, and creates object records for all but DC, CCW and literal statements. The input for this phase comes from SYSUT2 or SYSUT1 depending on the number of alternating passes made by Phase 10 and writing is done on the other of the two data sets.

On completion of this phase, control passes to Phase 21B.

Functions

The functions of this phase are designed to perform the following:

- Complete the processing of all machine instruction statements.
- Evaluate all fixed-point (types F and H) and floating-point (types E and D) constants and place the evaluation in the evaluation work buckets (see Appendix E).
- Write source and error records unchanged into the text stream interspersed with the edited text and object records.

The input record is checked to determine its type, then processed accordingly.

Source and Error Records. Source and Error records are passed to Phase 21B unchanged.

Edited Text. Edited text is further checked to determine the statement type.

For machine operation statements, object records are created to be used in printing the left-hand side of the listing. The location counter value, machine instruction, operand addresses (in base displacement format), and the effective addresses representing the operand fields, are entered into the object record. The Register Availability (Using) Records are used in deriving base-displacement addresses from effective addresses. The object records are placed in the output stream in place of the edited text records. For CNOP statements, the correct number of no-operation instructions is generated and placed in an object record. Statements containing type F, H, E, or D constants have their operands converted to binary and placed in the proper work buckets.

Statements other than machine instructions, CNOPs, or type F, H, E, or D constants are passed to the next phase unchanged.

Other Records. Records other than source, error, or edited text records are passed to the next phase unchanged (e.g., switches records, etc.).

In evaluating constants, evaluation work buckets (type H) are appended to the DC, and literal text records as required. If a DC or DS statement is flagged to not be evaluated, all the work buckets appended to the text, except one blank (type B) delimiter work bucket, are eliminated from the output of this phase (see Appendix E).

Routine Descriptions

DCNVRT

Processes exponent and scale modifiers.

ECNOP

Generates the required number of no-operation instructions and places them in an object work bucket.

GETPTR

Gets pointer of the next logical record in the input buffer.

GETXTM

Gets text and moves it for use by 21B/21C/21D.

LITRDC

Converts the operand field of type F, H, E, and D constants into binary.

ML1

Resets the error count and error switches, locates the latest input record, tests for end-of-data, and determines the record type.

ML2C

Passes source record to 21B.

ML2J

Outputs the current record.

ML2M

Points to the record to be output and branches to routine PUTERR which puts this record into the text output buffer.

ML3

Tests the 'catastrophy' bit and, if the bit is set, branches to ML1; otherwise, starts building the left-hand print area.

ML3A

Tests for CNOP to determine if alignment is needed.

ML3X

Branches to routine CTRLEV which evaluates the operand fields of machine operation instructions.

ML4T

Dumps a record containing an error into the text stream.

ML4X

Processes machine operations.

PHCLS

Phase closing routine.

PUTERR

Writes errors.

PHASE 21B/21C/21D - DECLARATIVE PROCESSING AND OUTPUT - CHARTS 28, 29, 30

The input for 21B is from SYSUT2 or SYSUT1. This program segment evaluates and converts declarative statements (DC/DS/Literals) to their object form and formats them for writing. Relocation List Dictionary segments for relocatable and V-type constants are created and written in 21B. The program segments 21C and 21D are alternately in main storage with 21B. The processing of DCs of type A, Y, V or S is done in segment 21D. All other address constant expressions including all CCWs are processed in 21C. The RLD table is written on SYSUT3. The object program is punched on SYSPUNCH and the program listing is printed on SYSPRINT.

On completion of this phase, control passes to Phase PP.

Functions

The functions of the phase are designed to perform the following:

- Convert declarative statements to their object format, including any necessary truncation or padding.
- Decompose any S-type constants into base-displacement form.
- Evaluate type A, Y, and S constants which contain location counter references (*) and have a duplication factor greater than one (overlay 21D).
- Build a Relocation List Dictionary Table containing entries (see Appendix I) for relocatable type A, Y, and V constants, and relocatable second operands of CCW statements.
- Write the RLD Table on SYSUT3 for processing by Phase DPP.
- Pass error records to Phase DD.
- Punch the object program, in object module form, on SYSPUNCH.
- Print the program listing (excluding RLD, cross-reference listing, and error messages) on SYSPRINT.

The current input record type is checked to determine if it is a source record, edited text, or some other type.

Source Records. Source records are used to form the right hand side of a listing line.

Edited Text. Edited text is checked to determine the statement type. The statement may be an MNOTE, PUNCH/REPRO, DC/DS, CCW, or a literal.
MNOTE - reset switches and bring in a new record.
PUNCH/REPRO - translate and write on SYSPRINT and SYSPUNCH.
CCW - the operand is evaluated, formatted and written.
DC/DS - perform any necessary alignment, evaluate the operands, and write the results each time eight bytes have been accumulated. If the PRINT 'DATA' option is not specified, only the first eight bytes of the evaluated constant will appear on the listing. In evaluating constants, one of two evaluation routines will be used. The deciding factor is the combination of parameters used in the operand fields. Any type A, Y, or S constant that contains a location counter reference and has a duplication factor greater than one, must be evaluated by a routine contained in overlay 21D. Other constant operands are evaluated by a routine in overlay 21C, which is considered to be the normal overlay. Each DC/DS operand has its own work bucket from which the required evaluation routine can be determined. Routine SNFRD checks whether the required evaluation routine is in core and, if it is not, branches out to either OVERLAYA or OVERLAYB to bring it in.

It is possible that both evaluation routines may be used in evaluating all of the operands of a statement, and either one could be the last one brought into main storage for a given statement. Since 21C contains a CCW evaluation routine which will be required if the next statement should happen to be a CCW, 21C is always left in main storage or brought in, if necessary, after the last operand of a statement has been evaluated.

Other Types of Records. Other types of records that might be encountered are object records (used to form the left-hand side of the listing), the dummy LTORG record associated with the Assembler END statement, register availability (USING) records, listing control switches record, and TITLE reformatted edited text records.

Information in the object records is

written on SYSPRINT and SYSPUNCH.

Availability, switches, and TITLE records are maintained in their appropriate table or storage area and appropriate listing and forms control action occurs. When an end-of-data condition is reached, a branch occurs to PHCLS which empties the punch buffer, puts an end-of-file indication in the Relocation List Dictionary and I/O Parameter Table, releases the 400 bytes of storage acquired for this phase, and passes control to Phase DPP.

Routine Descriptions

BLIGN

Perform any necessary location counter alignment.

BMG

Processes the first operand.

CHKSWJ

Entry point to the print routine.

DCEVAL

Evaluates DC and DS statement operands using subroutines SNFRD, BLIGN, DSCHECK, AY, CT, SUB1.

DUMP

Entry point to the punch routine.

EIN8

Processes the first operand.

EVLCCW

Processes CCWs.

F8PRNT

Output routine. Includes CHKSWH, LOADLH, LOADRA, GOTXT.

LOADRA - load right half of output page

LOADLH - load left half of output page

CHKSWH - print routine

GOTXT - punch routine

GETBKT

Gets the next work bucket.

GETPTR

Gets the pointer to the next logical record in the input buffer.

LBJ

Brings in evaluation routine.

LINK21C

Calls overlay 21C.

ML7

Phase initialization.

ML9

Determines the record type and branches to the appropriate handling routine.

ML9M

Moves data to Value and Availability Tables.

ML9X

Punches TXT cards.

ML10

Sets up the heading.

ML11

Determines the type of record.

ML12

Picks up the ESD-ID number.

ML13

Prints the record.

ML13X

Moves in the processed left side of the listing.

ML14

Moves processed left side of listing for printing.

ML15

Tests for error record.

ML20

Writes error.

ML30

Checks for MNOTE error.

OVERLAYA

Calls in overlay 21C.

OVERLAYB

Calls in overlay 21D.

PHCLS

Phase closing routine.

PHSIN

Phase initialization routine.

PUTERR

Writes the error record.

SNFRD

Finds the DC work bucket and moves the required information from the work bucket into aligned storage. Checks if the proper overlay is in core and brings it in if necessary.

SNFRDD

Processes entire DC or DS.

WROF

Writes the overflow file entry point.

WRRLD

Writes overflow file.

ZWE13

Prints statement.

ZWE14

Processes DC.

PHASE PP - POST PROCESSOR - CHART 31

Phase PP reads RLDs and XREFs (cross-reference records) from the overflow file (SYSUT3). The RLDs are written on SYSPRINT and are packed into card images and written on SYSPUNCH. XREFs are sorted in core if possible, if not, a tape merge sort is performed using SYSUT2 and SYSUT1. The sorted cross-references are written on SYSPRINT. Phase PP also writes the END card on SYSPUNCH. All output from PP is controlled by the OPTION card.

Functions

The functions of this phase are designed to perform the following:

- Produce the Relocation List Dictionary and the Cross-Reference List, if requested.

Phase PP sorts the Relocation List Dictionary entries by address. The sorted dictionary is written onto SYSPRINT and SYSPUNCH. The loader END record is constructed and written onto SYSPUNCH.

If the EXEC record contained the option XREF, Phase DPP sorts the Cross-Reference Table entries by symbol. The sorted table is written on SYSPRINT.

Phase PP closes the utility units SYSUT2 and SYSUT1 and, using XCTL, transfers control to Phase DI.

NOTE: The algorithm used in the internal sort of Phase PP is that described by D. L. Shell in the July, 1959 issue of the ACM Communications (vol. 2, no. 7, pp. 30-32).

Routine Descriptions

CHKSWH

Sorts the Jump Table.

EPPGO

Initializes the phase.

EPRL2

Writes merge tape.

EP2

Writes merge tape.

ESORT

This routine performs the sorting of input data.

GTOR

Reads the Control Table.

GTOX

This routine checks for the Cross-Reference List output option.

PPIN

This routine initializes the Phase PP.

RDlRLD

Merges RLDs records.

RDlXRF

Merges cross-reference records.

READ

This routine reads the Relocation Dictionary input and stores it.

READR

Reads RLD records from input tape and stores them.

READX

Reads cross-reference records from input tape and stores them.

SETOTl

This routine writes all Relocation Dictionary data from main storage.

SETOT2

This routine writes all Cross-Reference data from main storage.

WRlRLD

Writes RLD string on SYSUT3.

WRlXRF

Writes cross-reference on SYSUT3.

XRFLOD

Start of cross-reference input pass.

PHASE DI - DIAGNOSTIC - CHART 32

Phase DI reads the error records from the overflow file (SYSUT3). Table lookup of error numbers is performed to find the corresponding error message. The statement number and message number are converted to printable format and listed with the error message on SYSPRINT. Phase DI output is controlled by the OPTION card.

Functions

The input to this phase is the error records on SYSUT3. The functions of this phase are designed to perform the following:

- Write diagnostic messages.
- Accumulate and print the total number of error statements in the entire assembly.
- Process the highest severity code.

Before printing any error messages a check is made to determine if any relocatable Y-type constants have been used in the program. If any have been used, message 46 prints as a flag to the programmer. The limited addressing capability of the Y-type constant, due to being only two bytes long, could present problems if the program is run on a system with over 65,535 bytes of storage.

Following the flagging of Y-type constants, a check is made to determine if there are any error records to be flagged; if not, the word 'NO' is inserted into the 'STATEMENTS FLAGGED...' message, this message is printed, and the phase exits to RTB. If there are error records to be flagged, each error statement number is listed with an appropriate message identifying the error. A total of the number of statements flagged is accumulated and printed.

As each error message is processed, its severity code is checked. The highest severity code encountered is saved in Register 15. The phase exits to RTB.

Routine Descriptions

EDGO

Locates the error block count, tests the Y-type constant indicator, and if necessary, points to message 46 in preparation for printing the 'AT LEAST ONE RELOCATABLE V-TYPE CONSTANT...' message. If there are no relocatable Y-type constants in the pro-

gram, branch to ML00.

GETERR

Reads error record.

HCC

Stores, in Register 15, the highest severity code encountered.

ML00

Tests if there are any error records to be processed. If there are no error records to be flagged, the 'NO STATEMENTS FLAGGED...' message is built, then a branch to ML11 occurs to list the message. If there are error records to be listed, this routine exits to M41A.

ML01

Gets the next error record. If the last error record has been read, a branch occurs to ML10.

ML01A

Gets the error statement number and accumulates an error-statement total.

ML01B

Converts the error statement number into decimal for listing and points to the appropriate error message.

ML03

Converts the error message for listing and lists it.

ML05

Compares the error statement severity code to the highest severity code yet encountered. Saves the new severity code if it is higher than any previously encountered.

ML10

Prints the total number of statements flagged.

ML11

Lists error message.

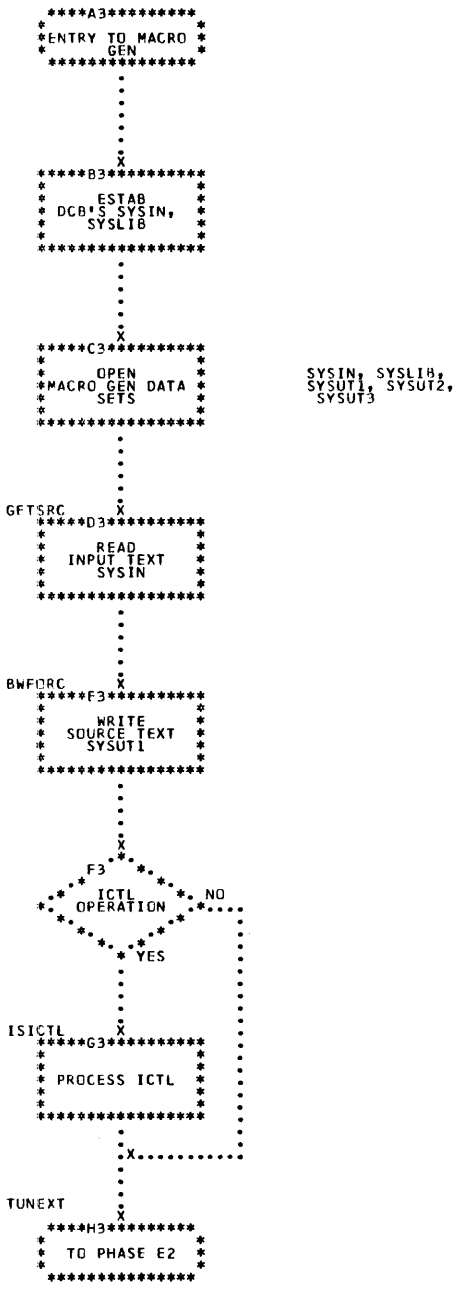


Chart 01. IETE1

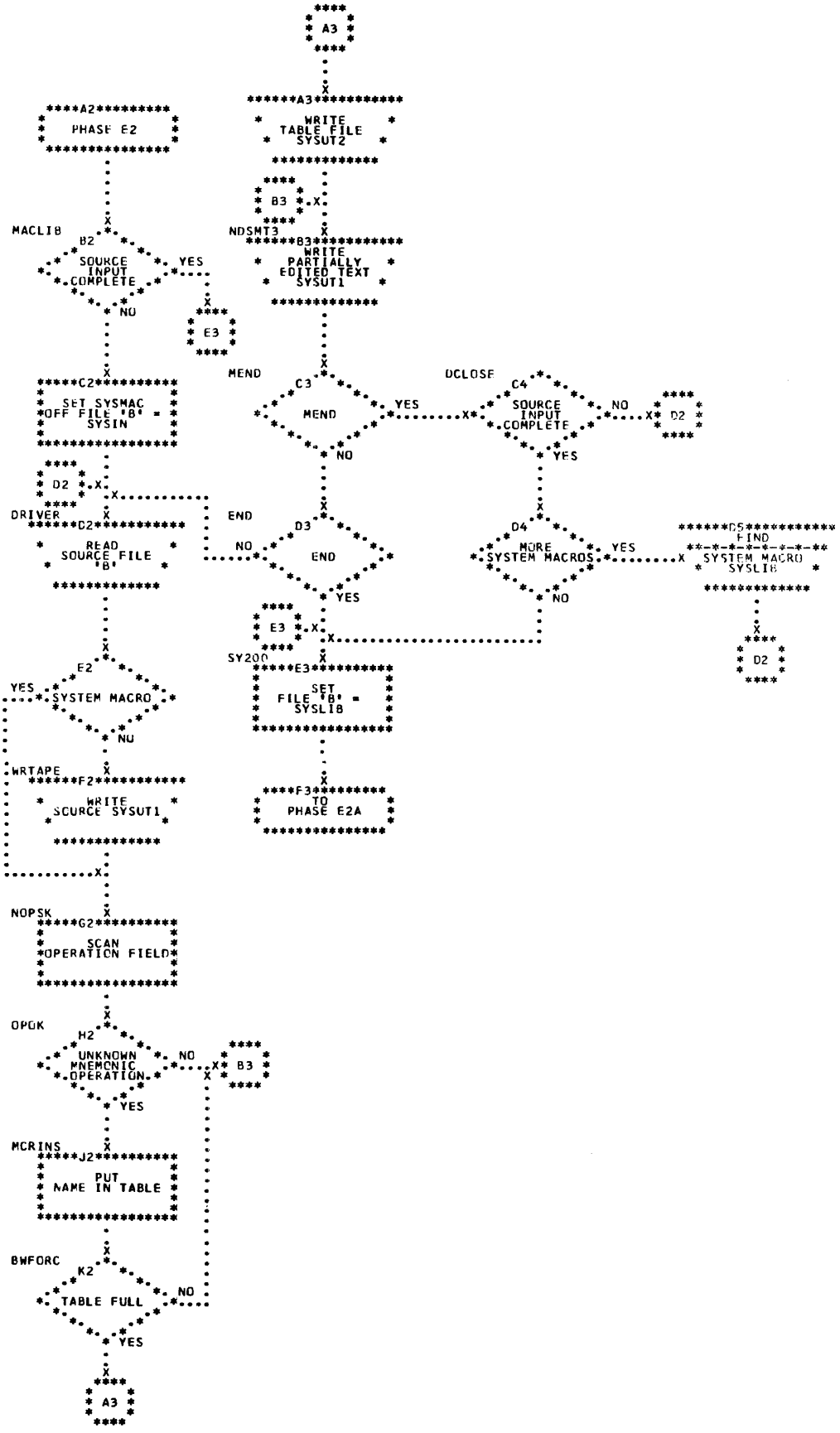


Chart 02. IETE2

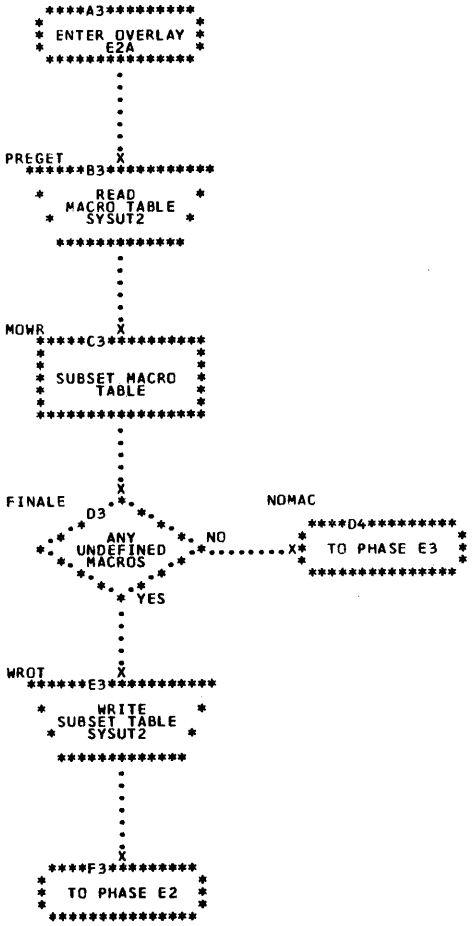


Chart 03. IETE2A

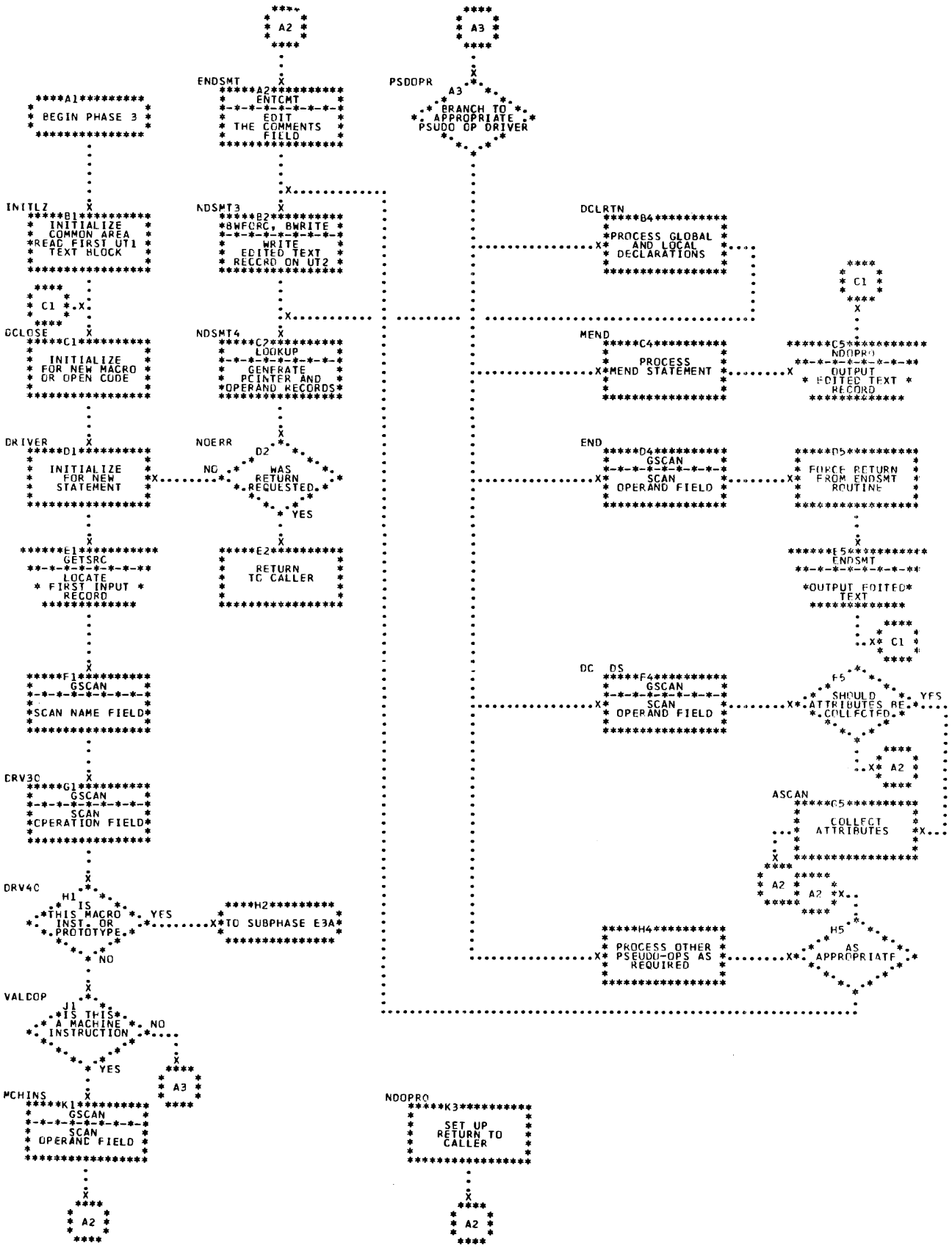


Chart 04. IETE3

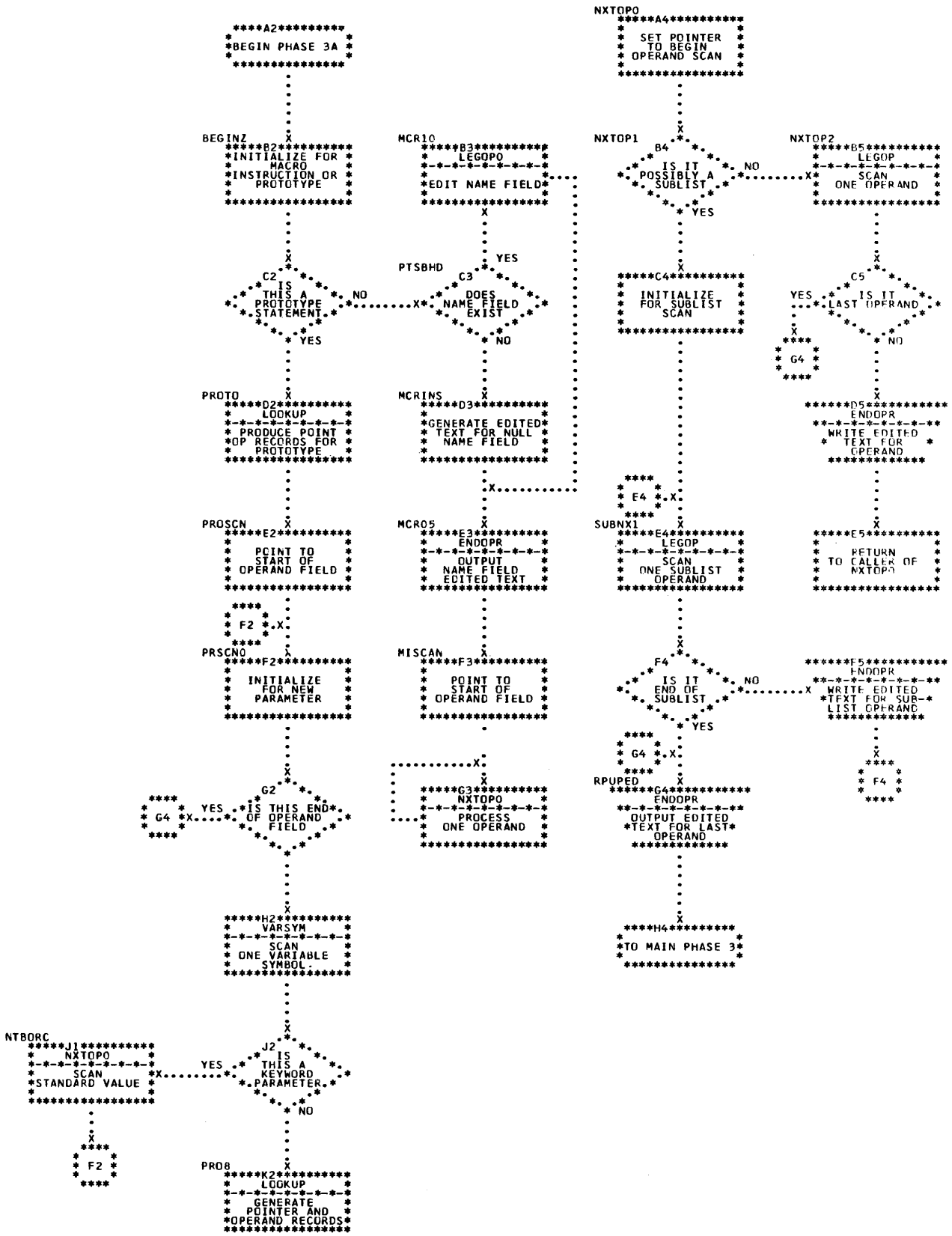


Chart 05. IETE3A

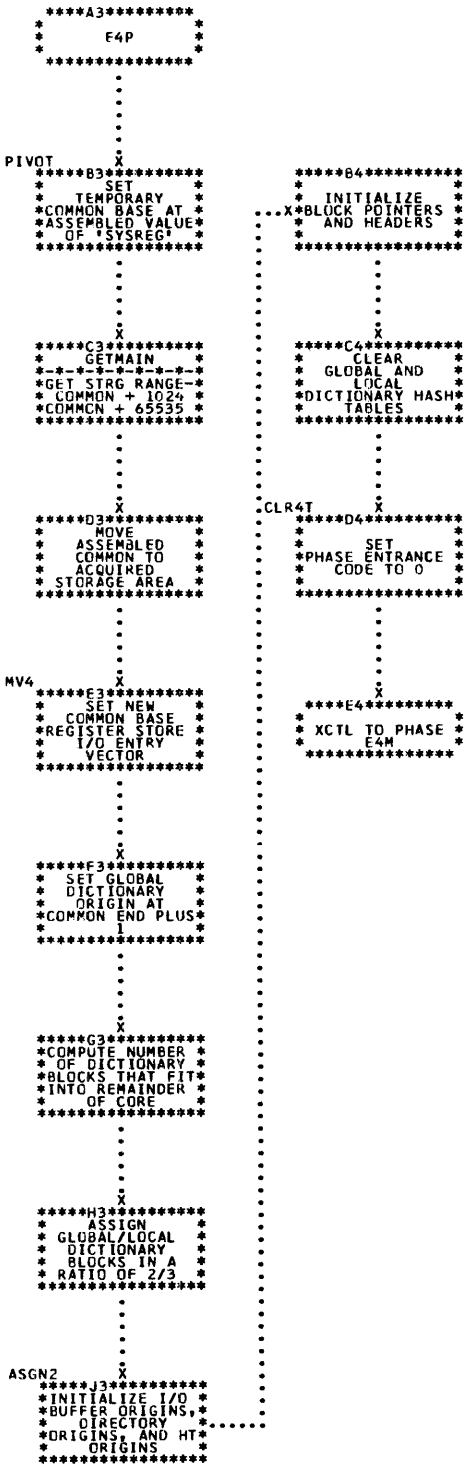


Chart 06. IETE4P

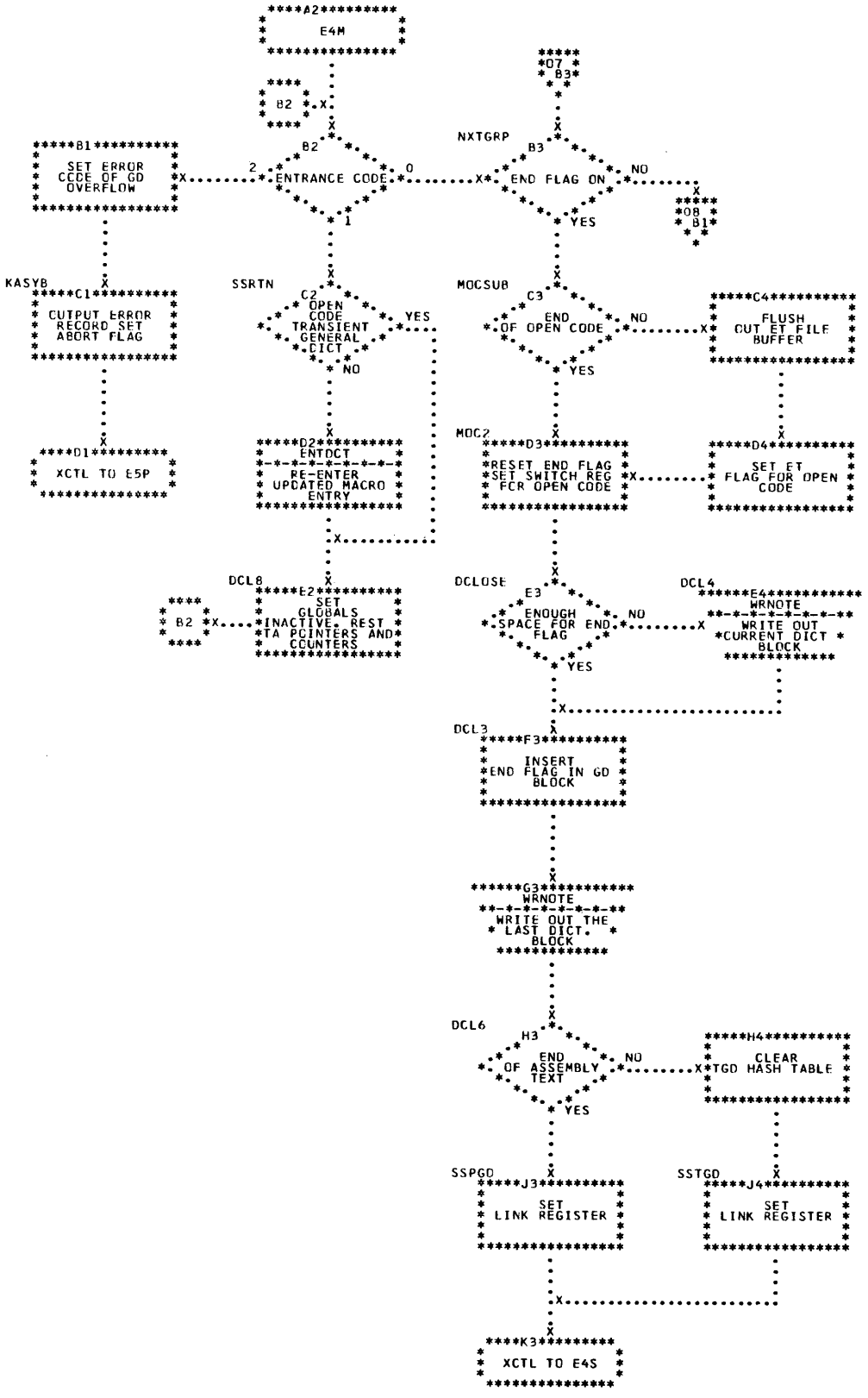


Chart 07. IETE4M

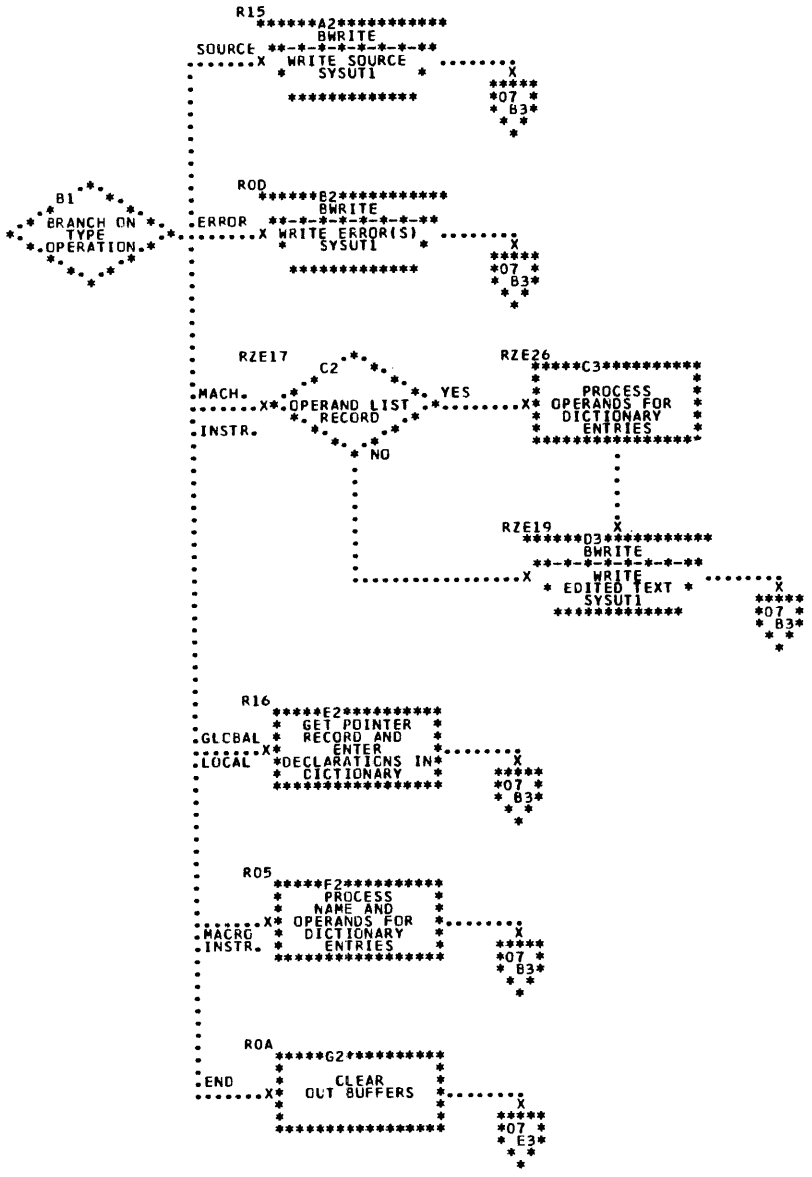


Chart 08. IETE4M

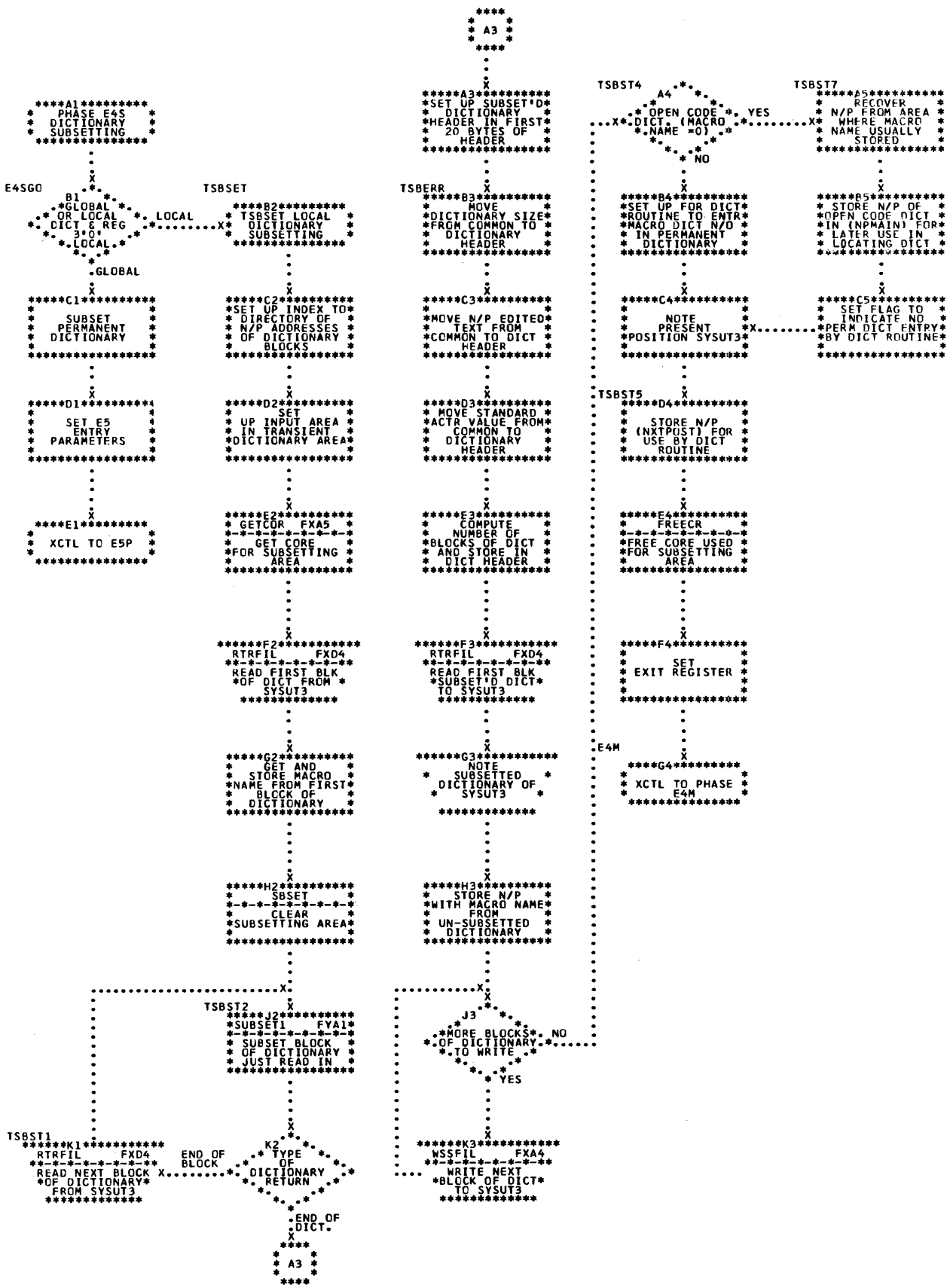


Chart 09. IETE4S

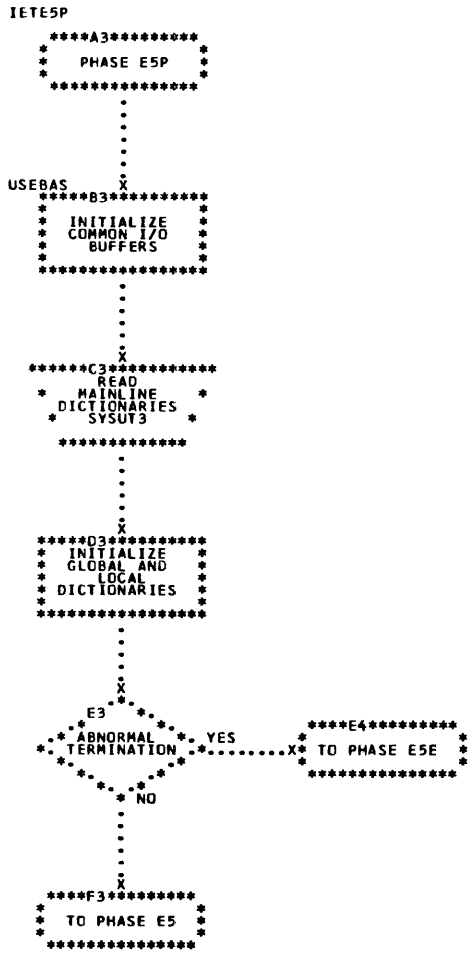


Chart 10. IETE5P

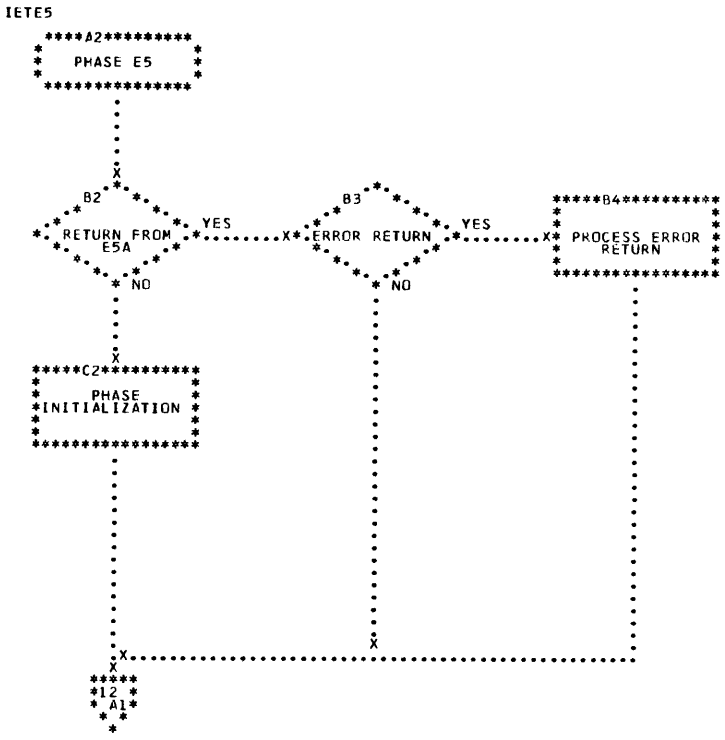


Chart 11. IETE5

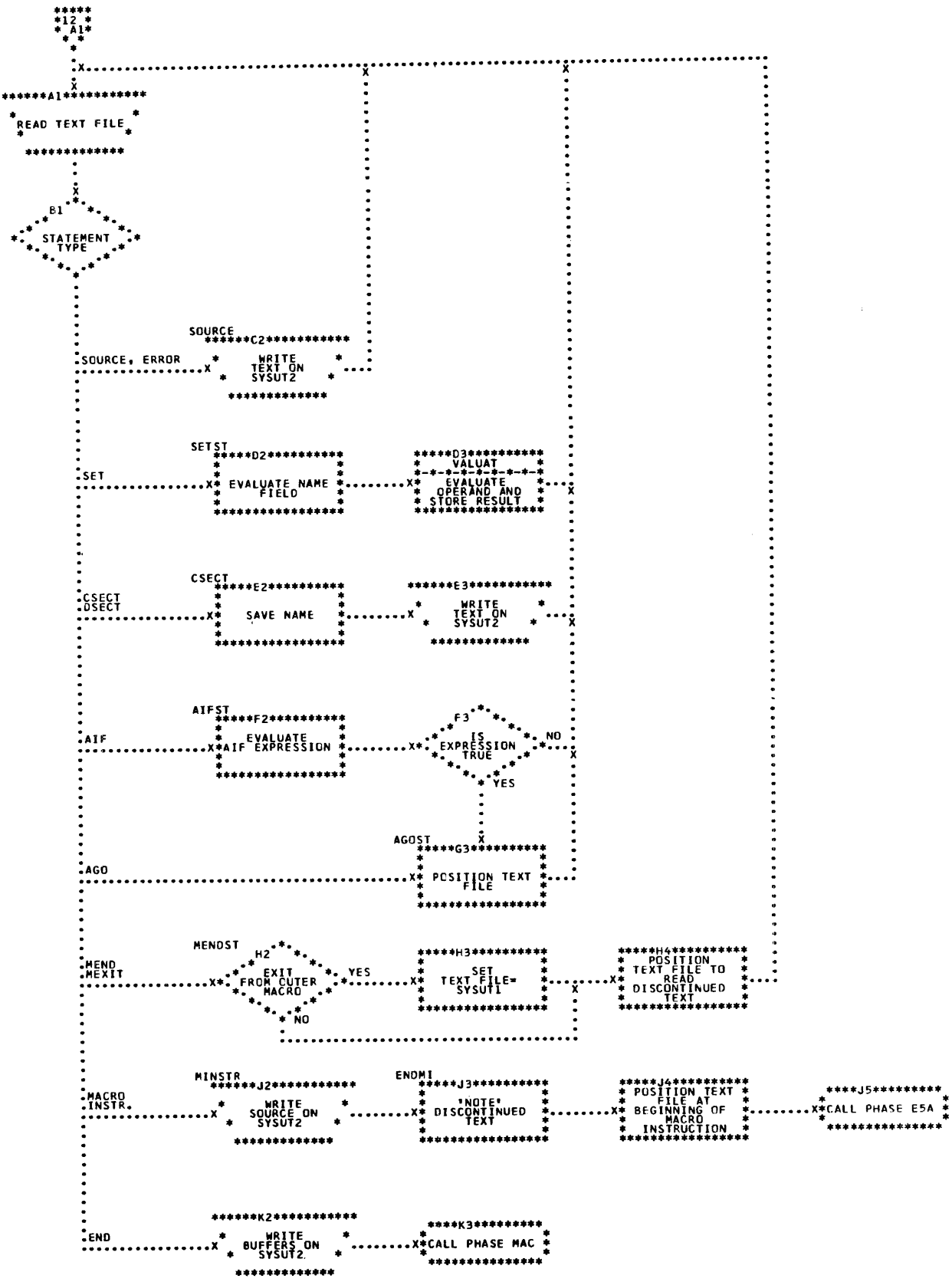


Chart 12. IETE5

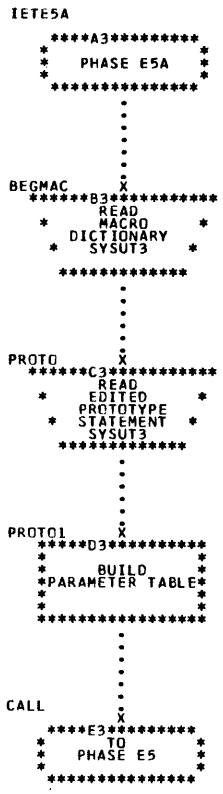


Chart 13. IETE5A

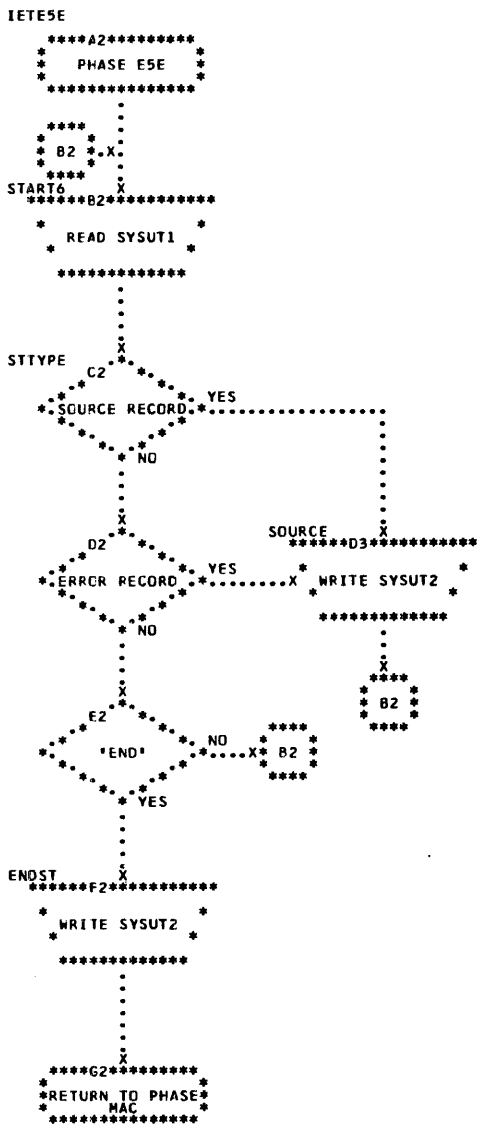


Chart 14. IETE5E

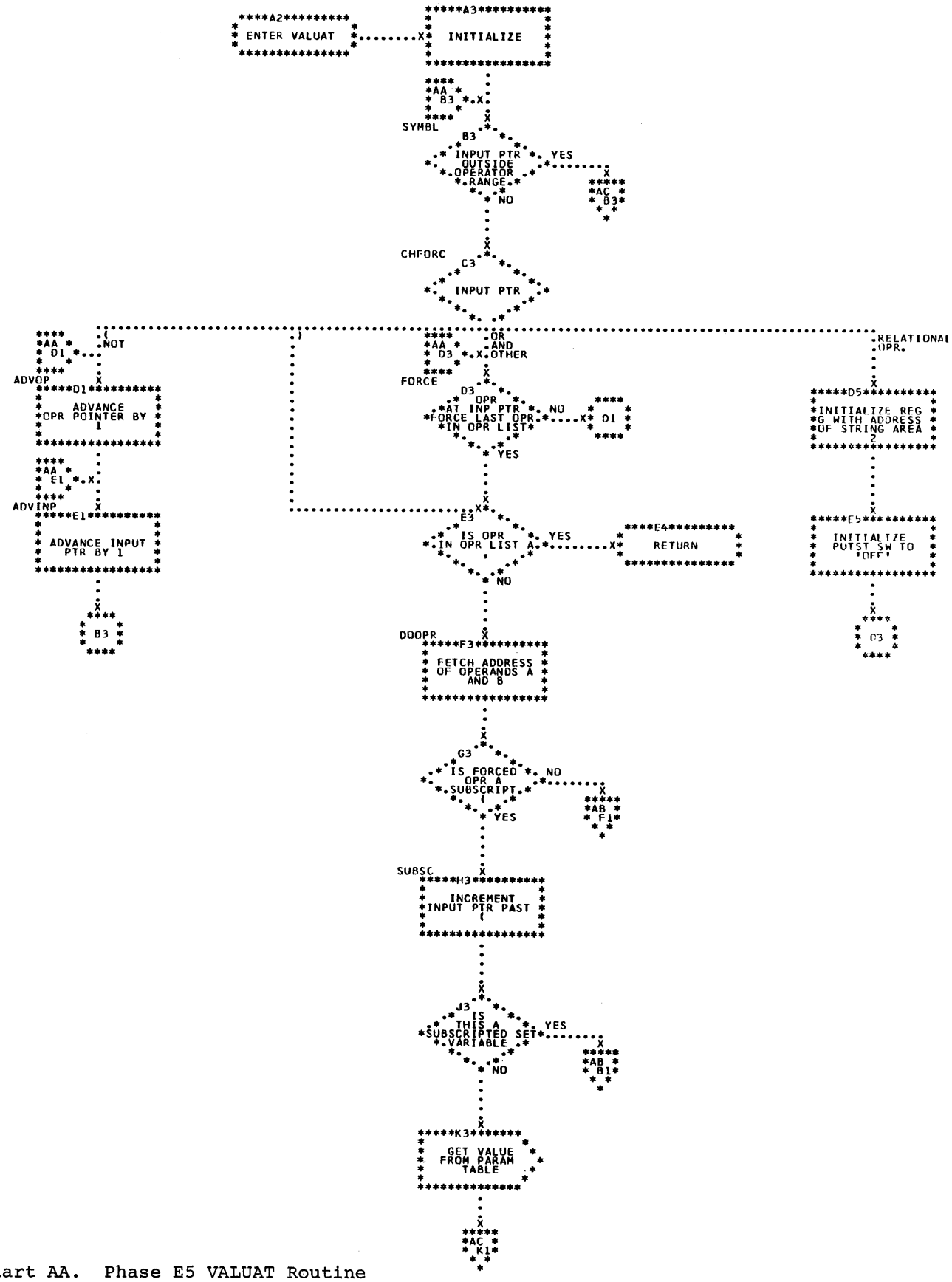


Chart AA. Phase E5 VALUAT Routine

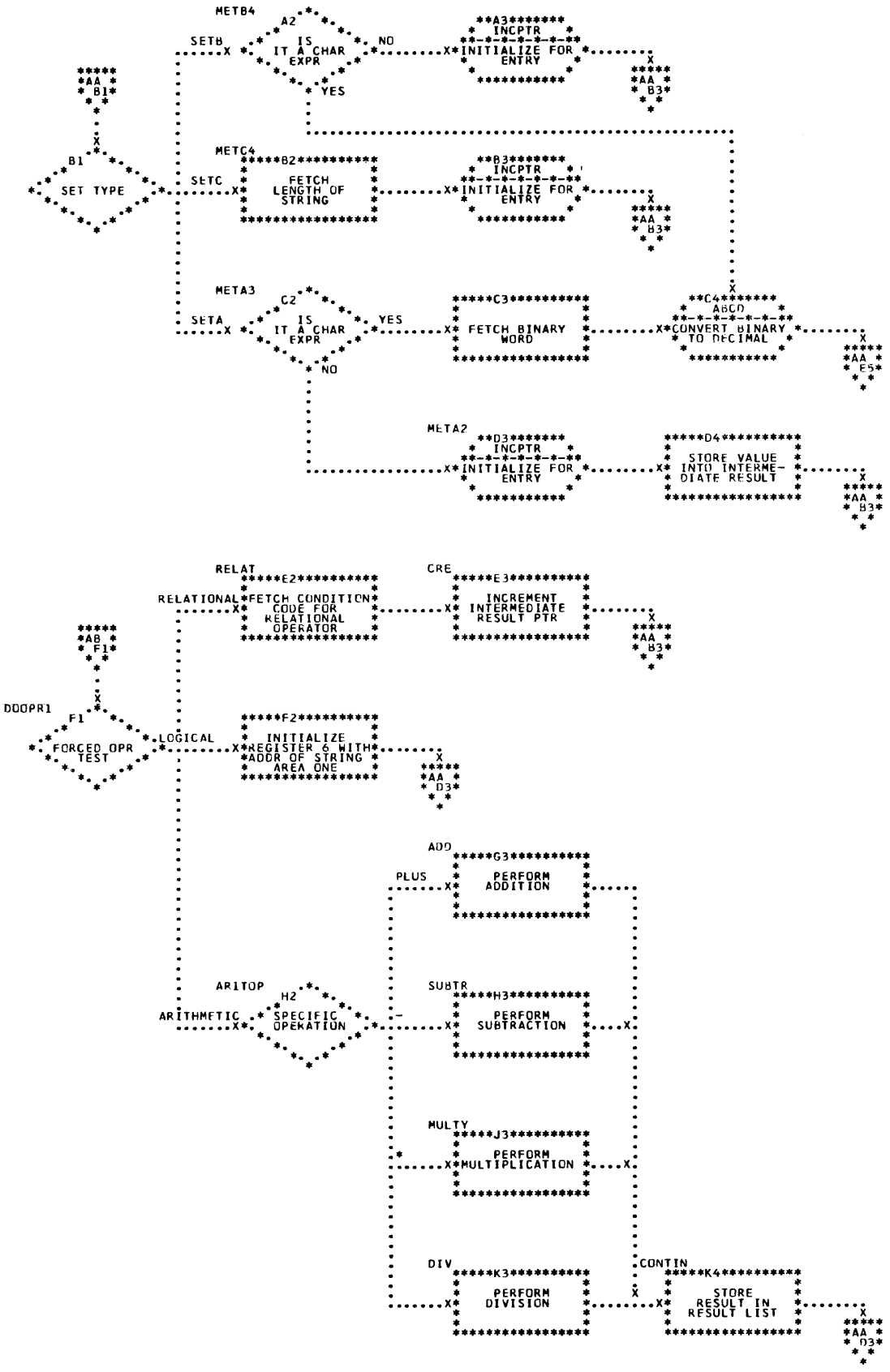


Chart AB. Phase E5 VALUAT Routine

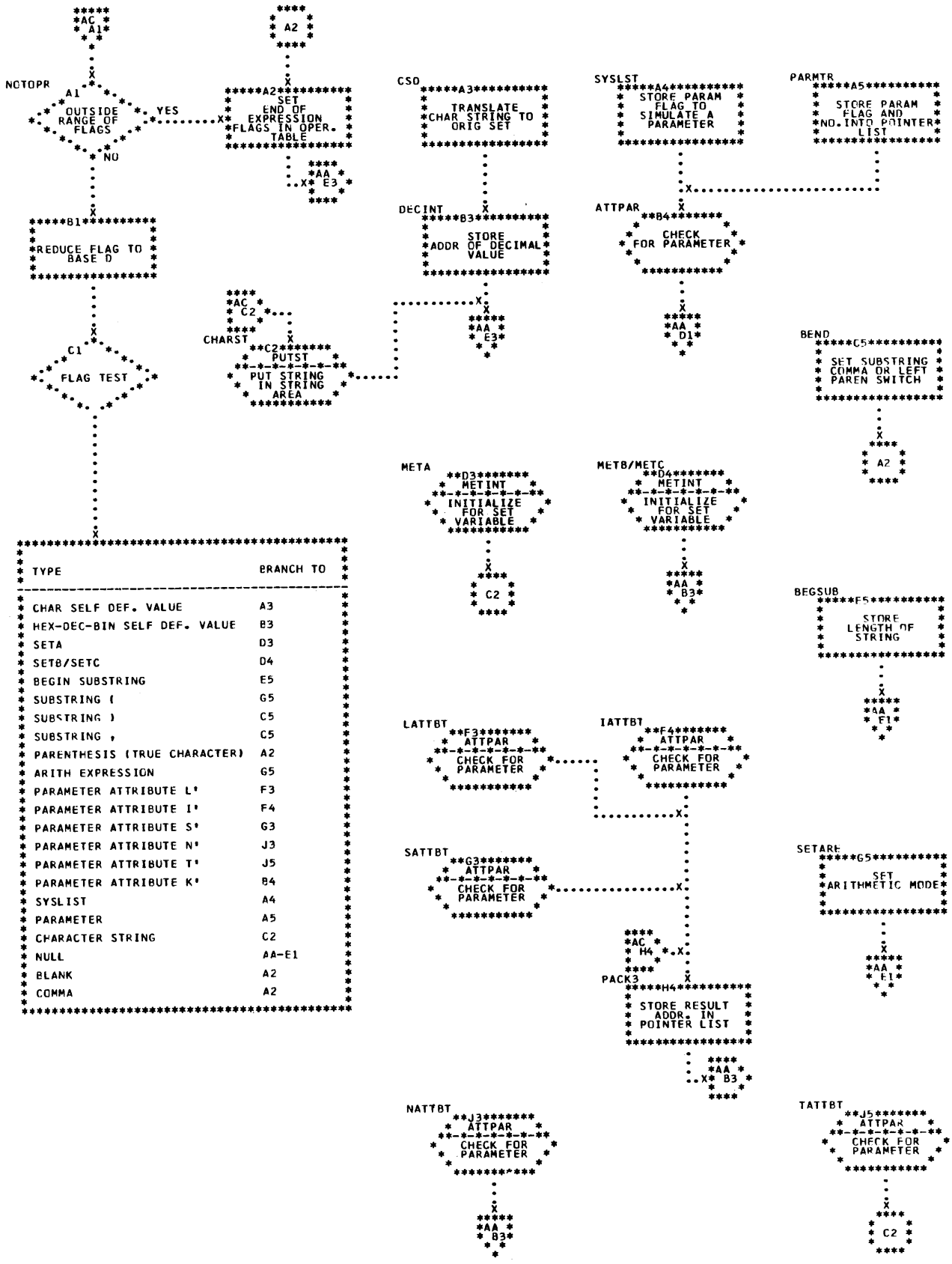


Chart AC. Phase E5 VALUAT Routine

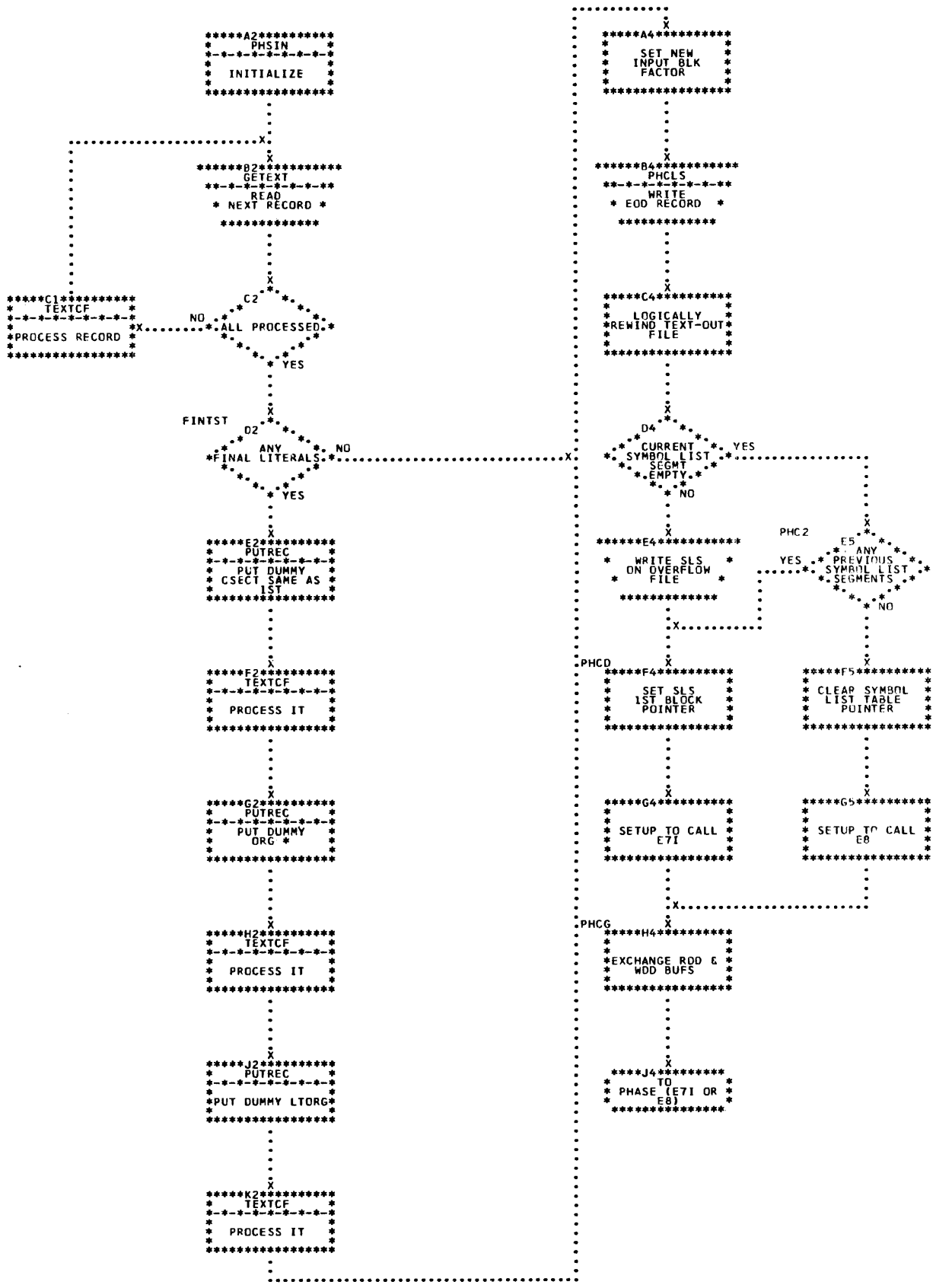


Chart 15. IET07/07A/07B

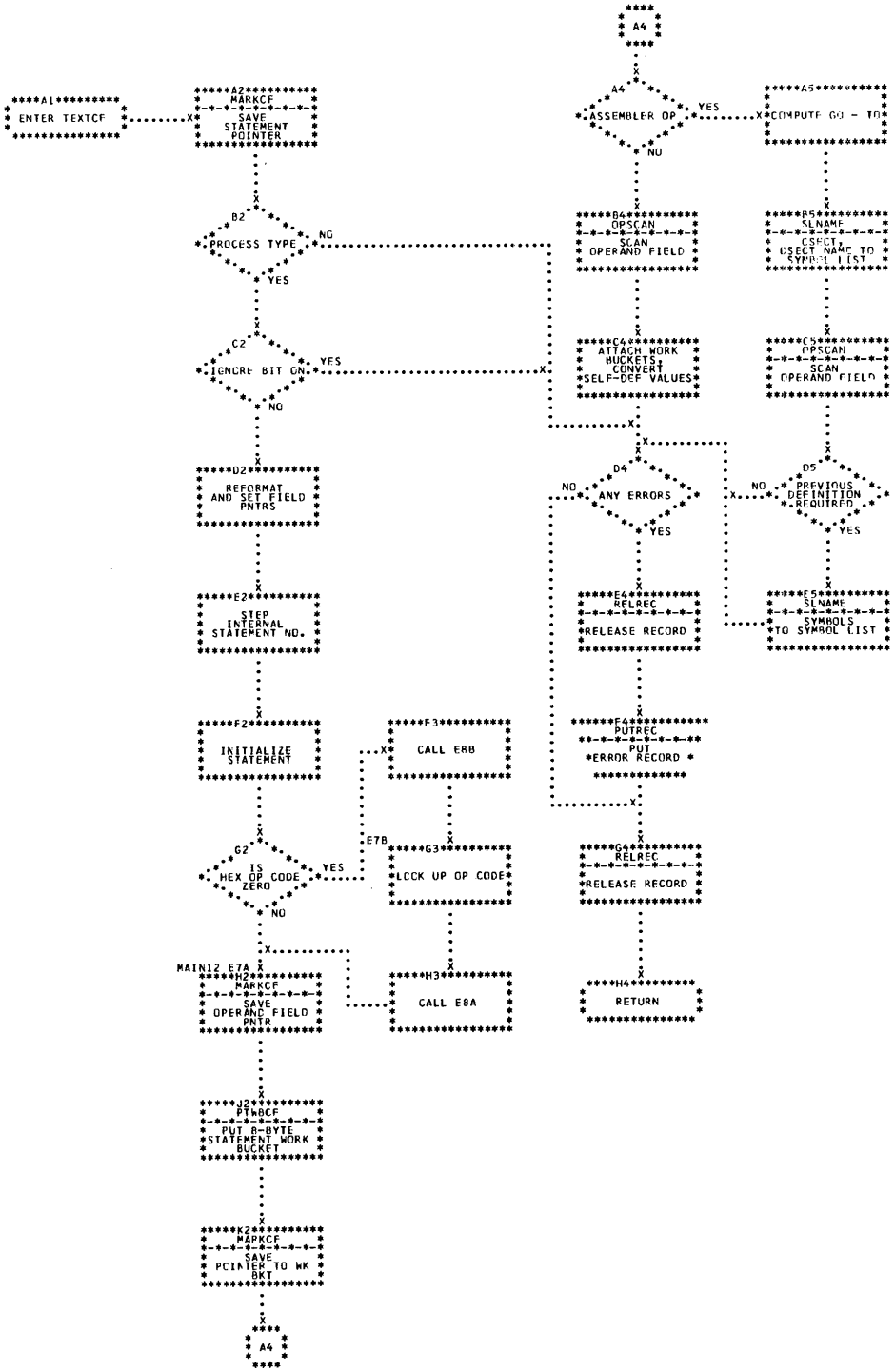


Chart 16. IET07/07A/07B

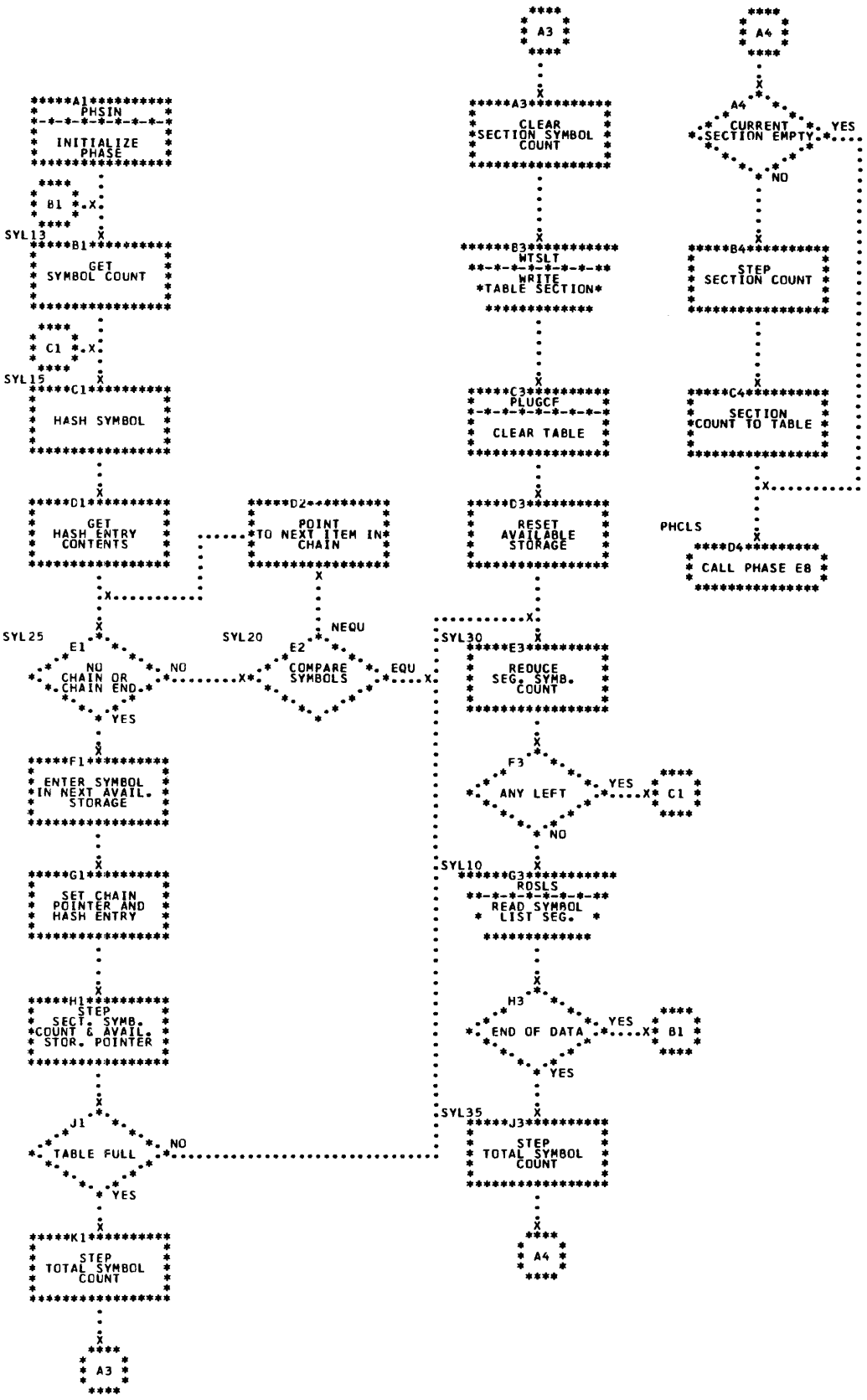


Chart 17. IET07I

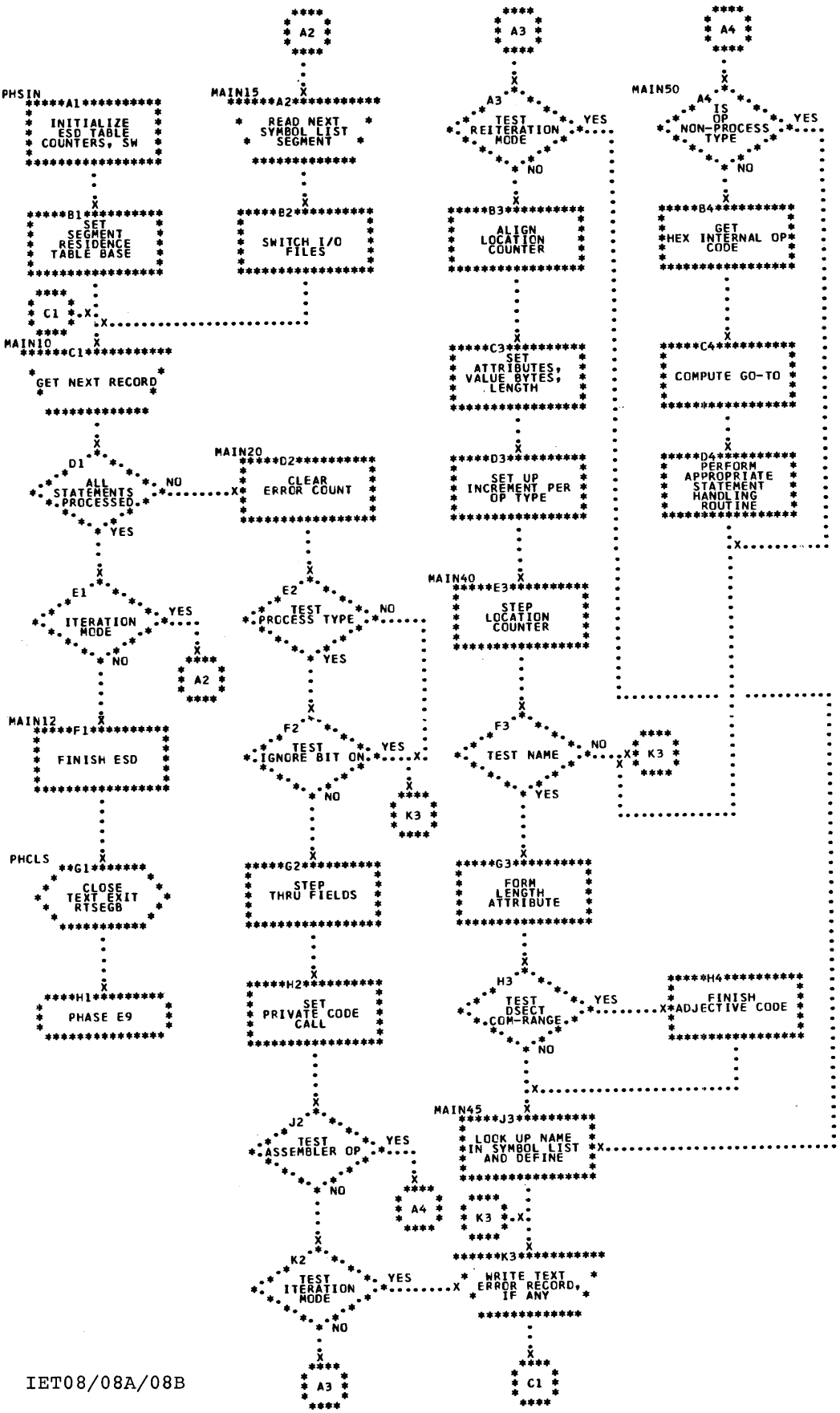


Chart 18. IET08/08A/08B

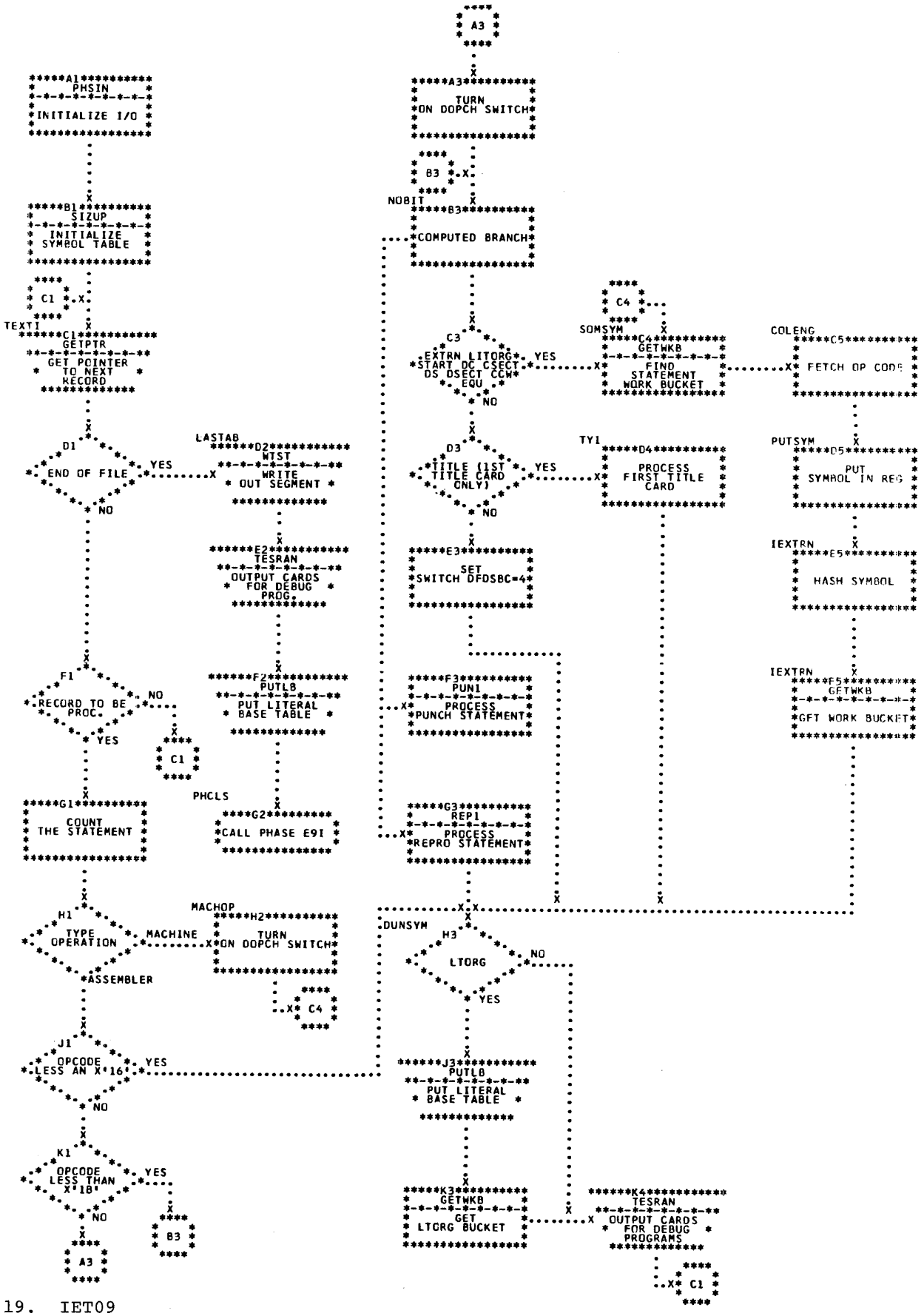


Chart 19. IET09

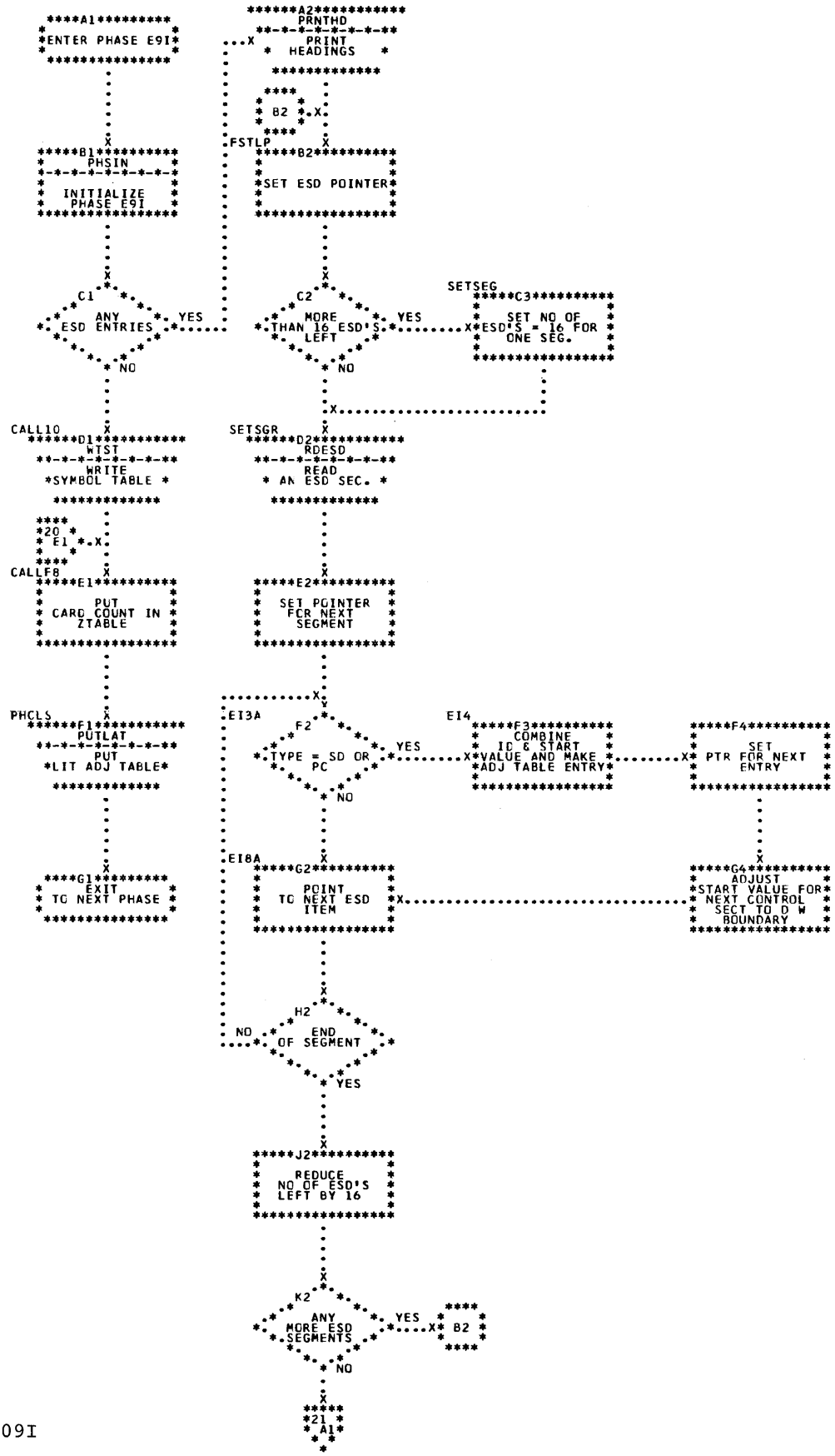


Chart 20. IET09I

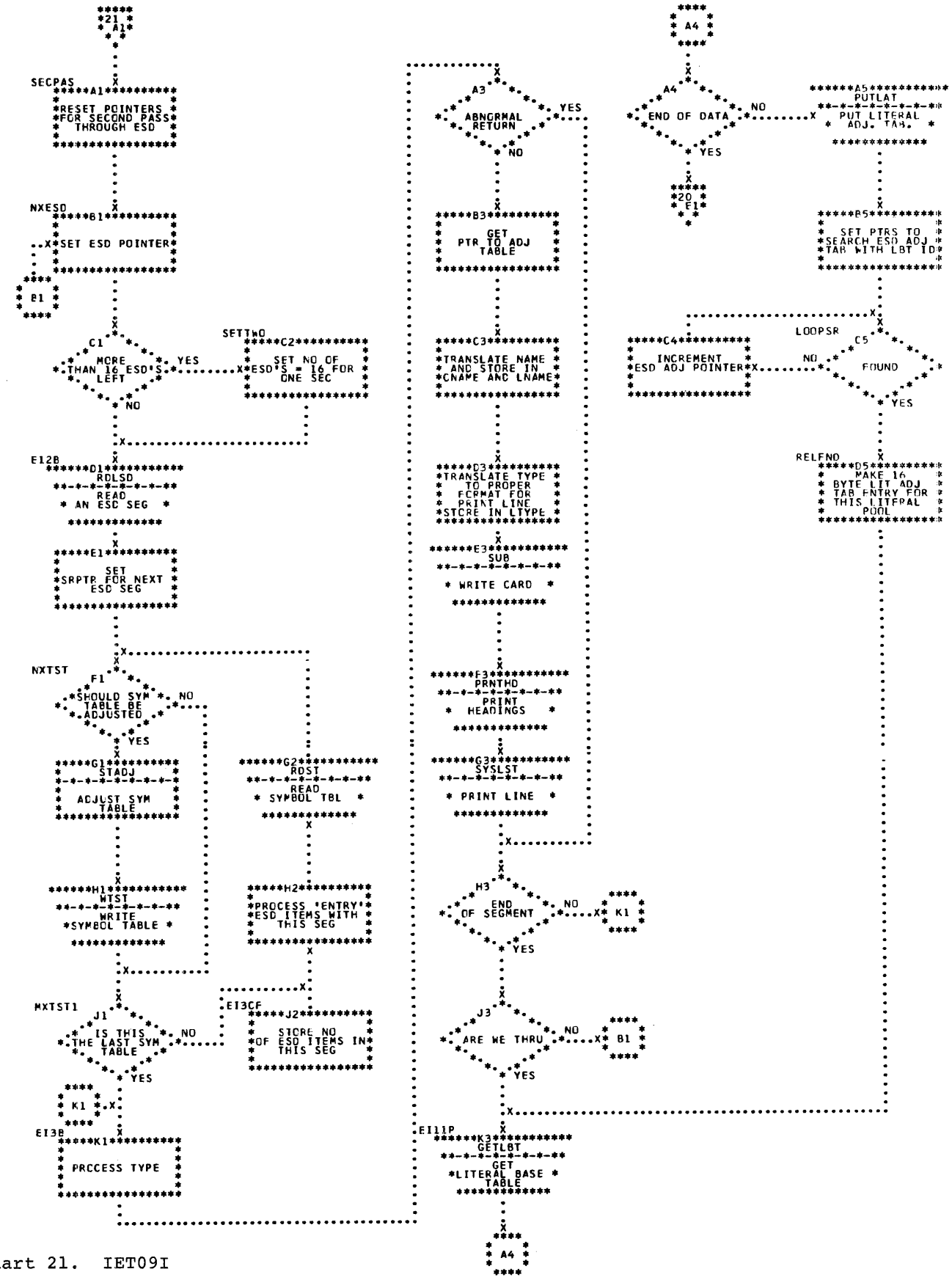


Chart 21. IET09I

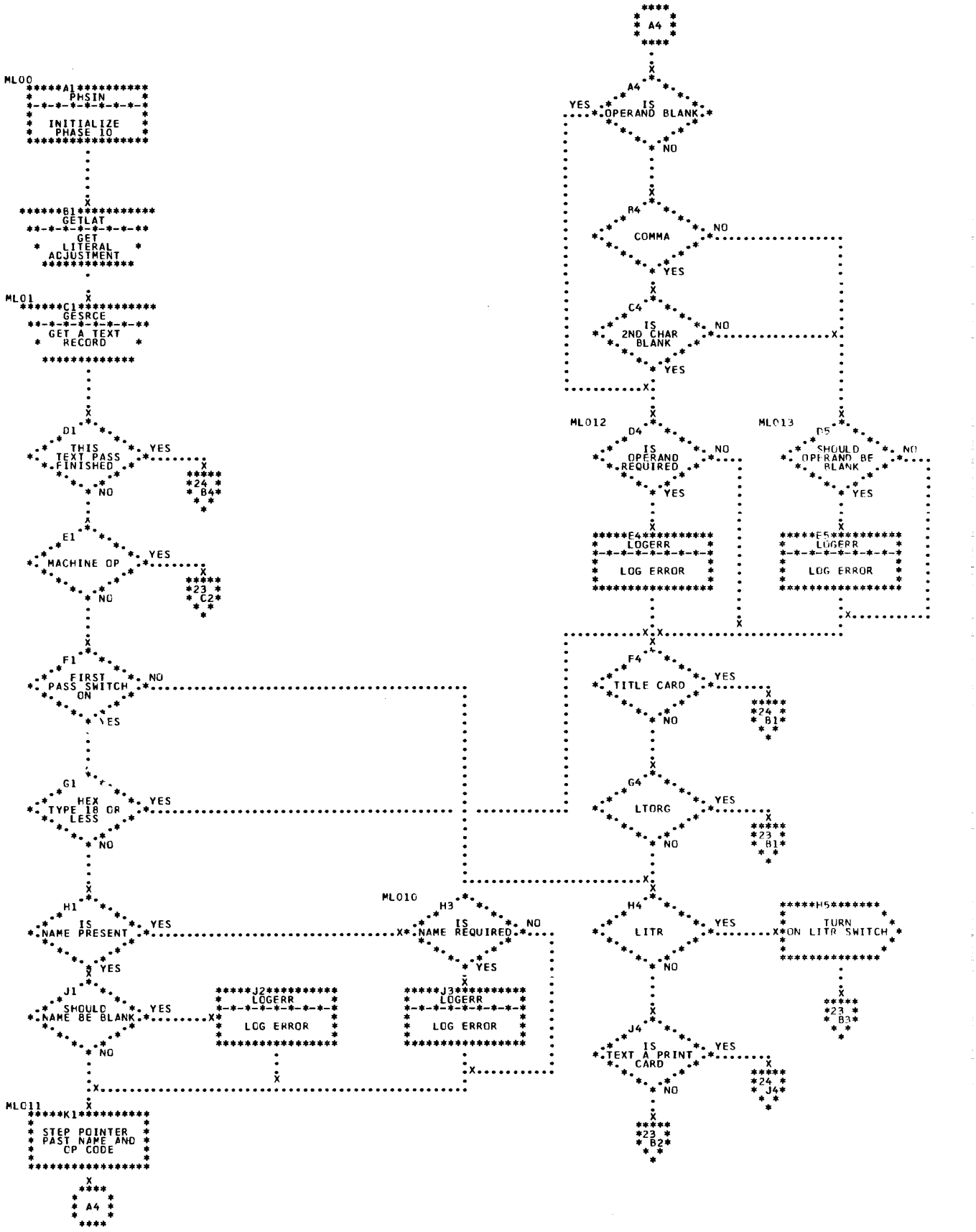


Chart 22. IETl0

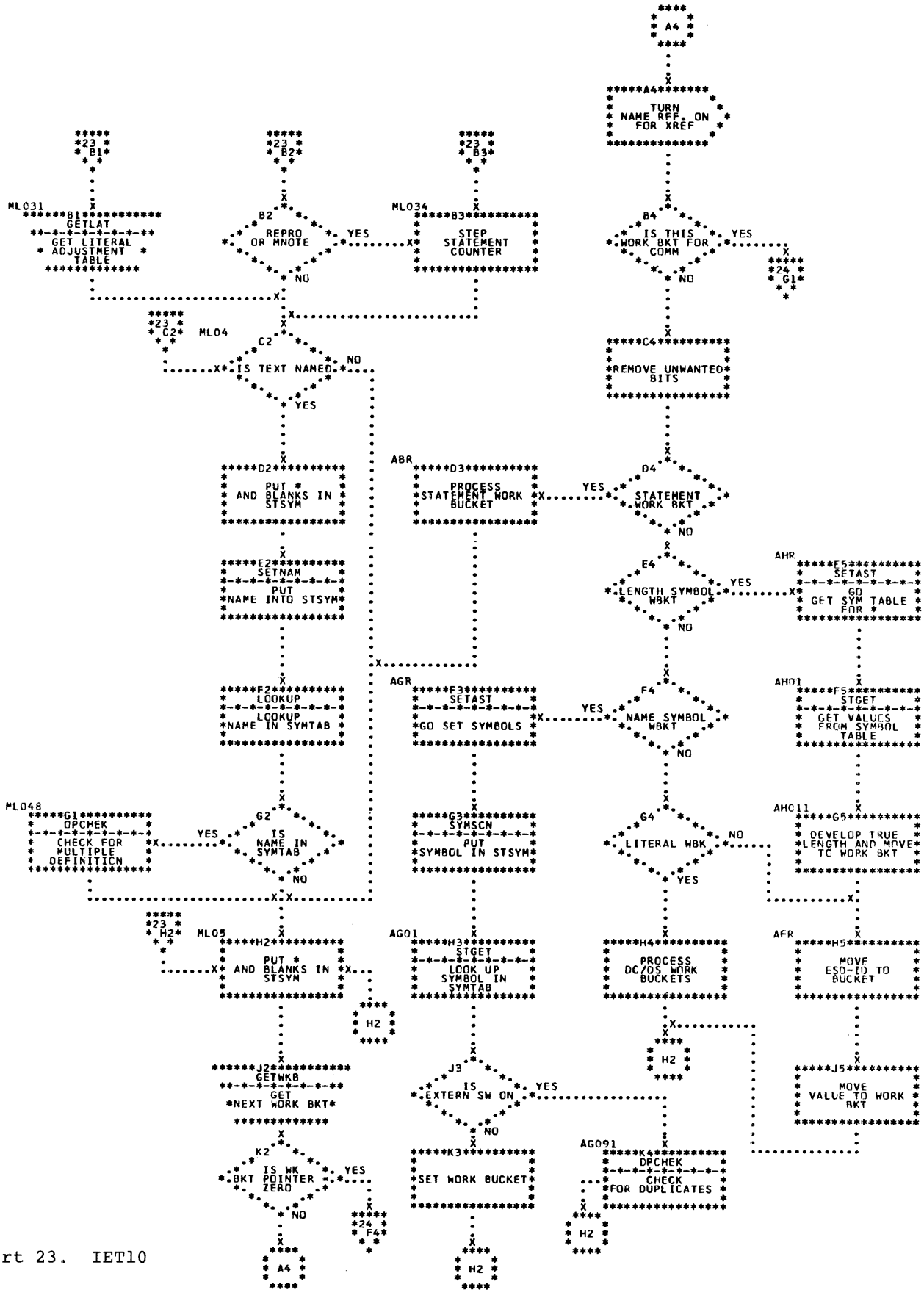


Chart 23. IETl0

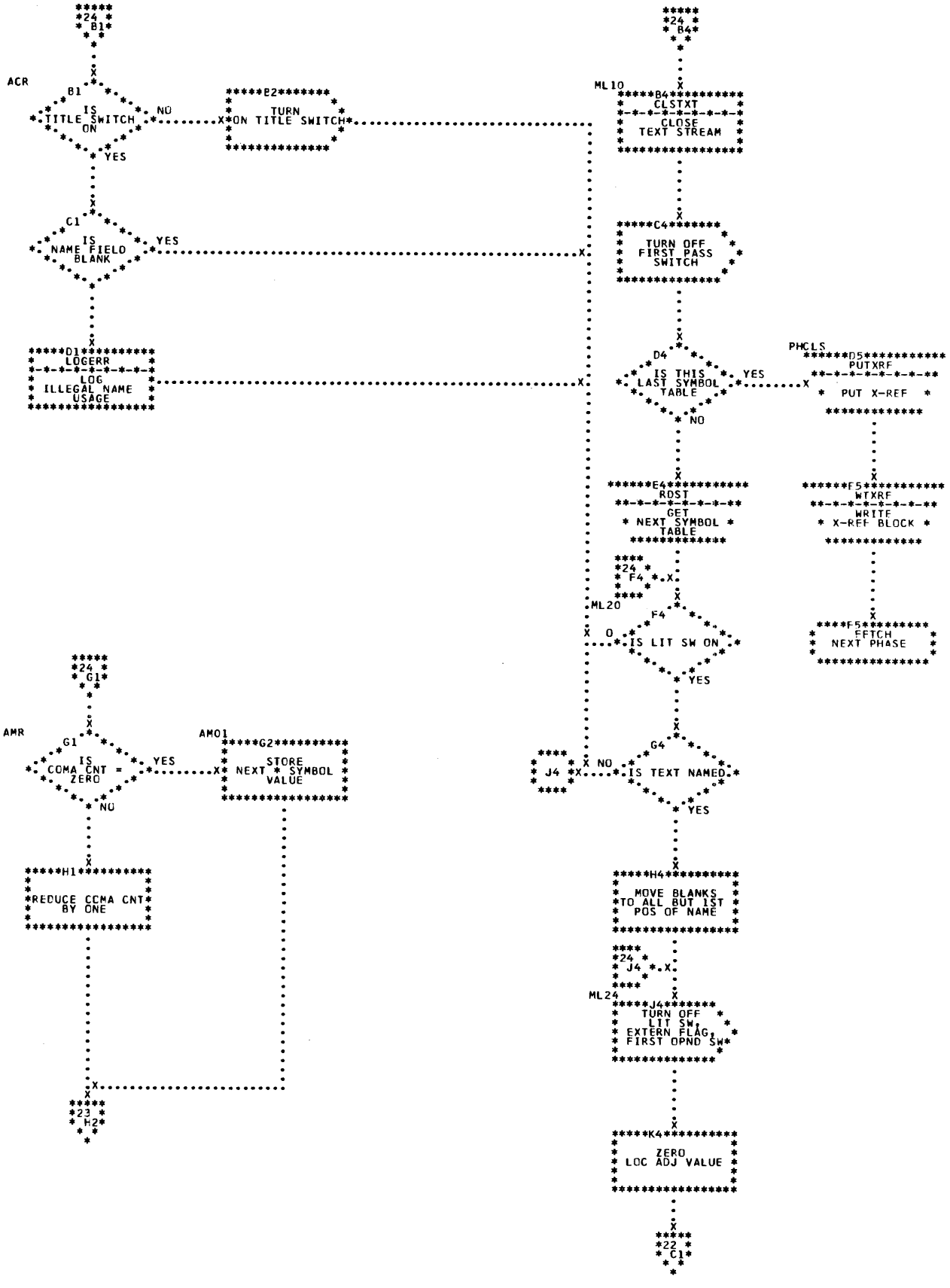


Chart 24. IET10

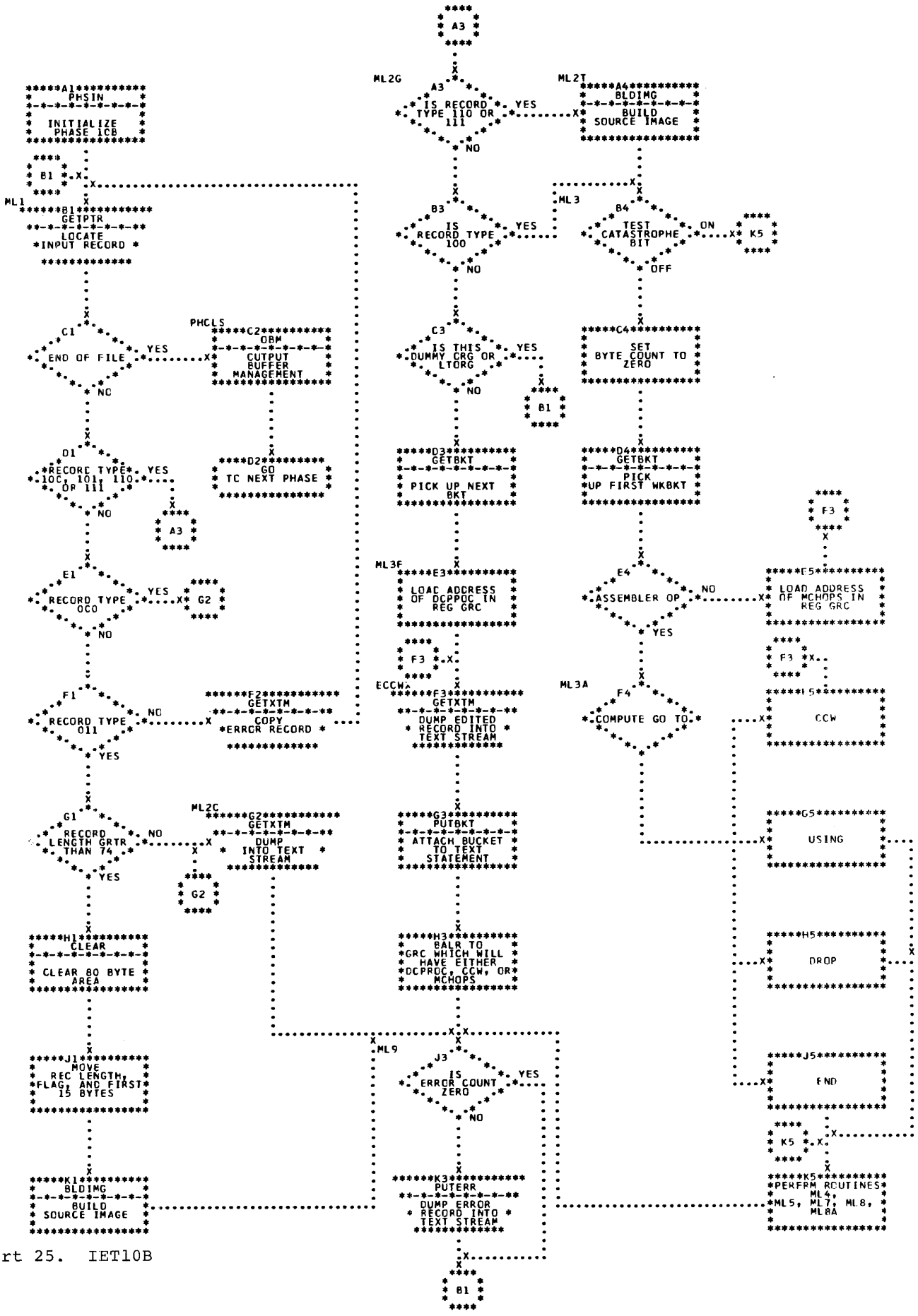


Chart 25. IETl0B

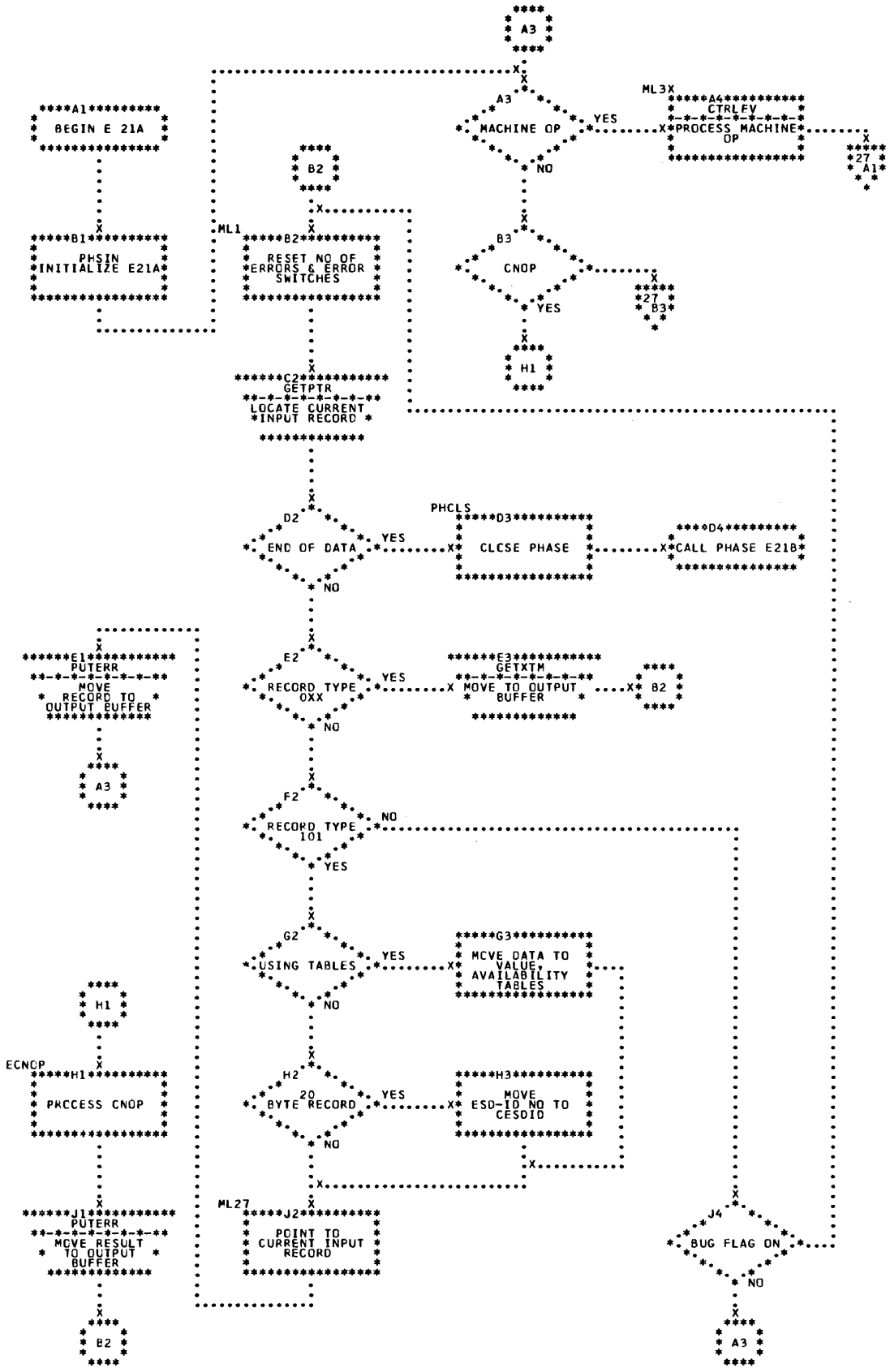


Chart 26. IET21A

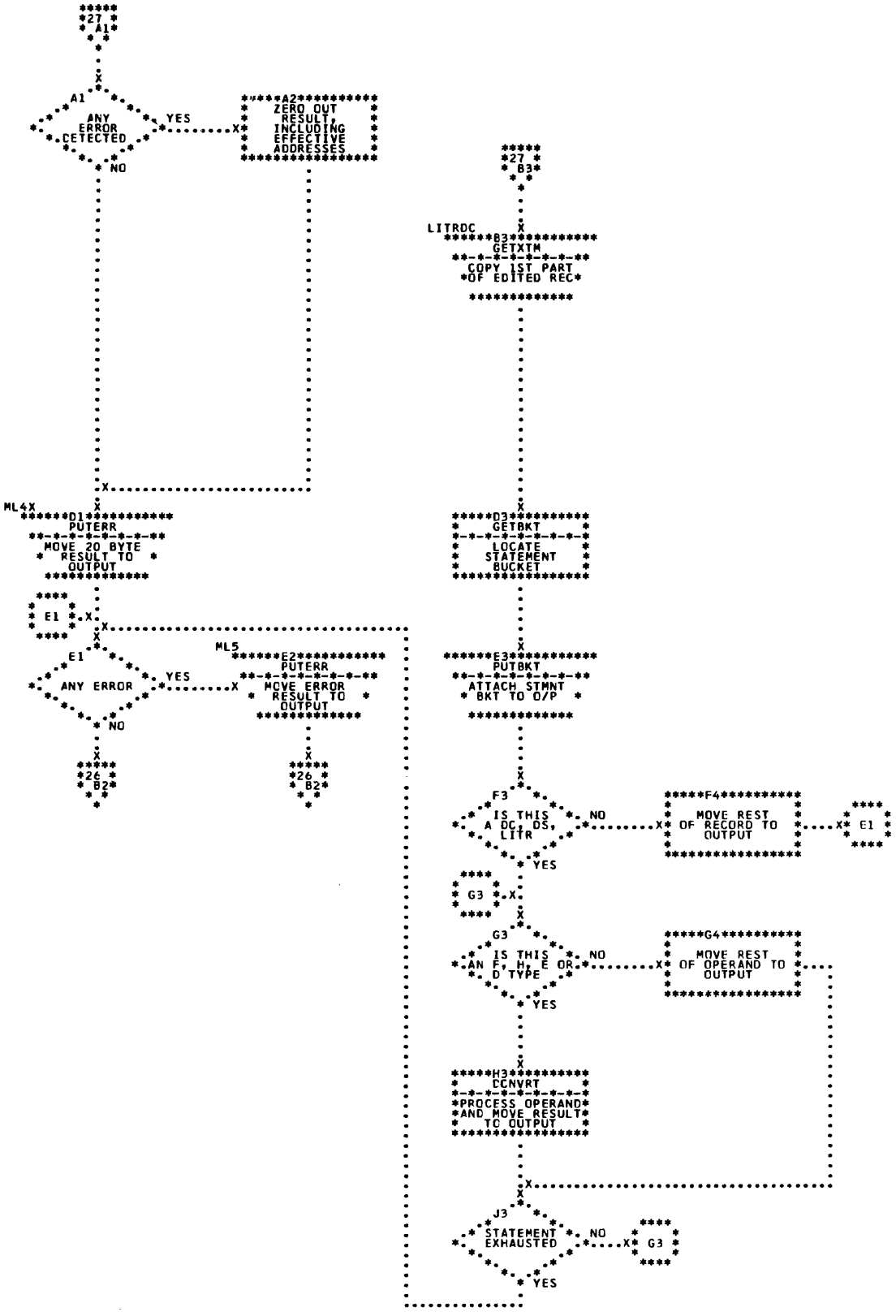


Chart 27. IET21A

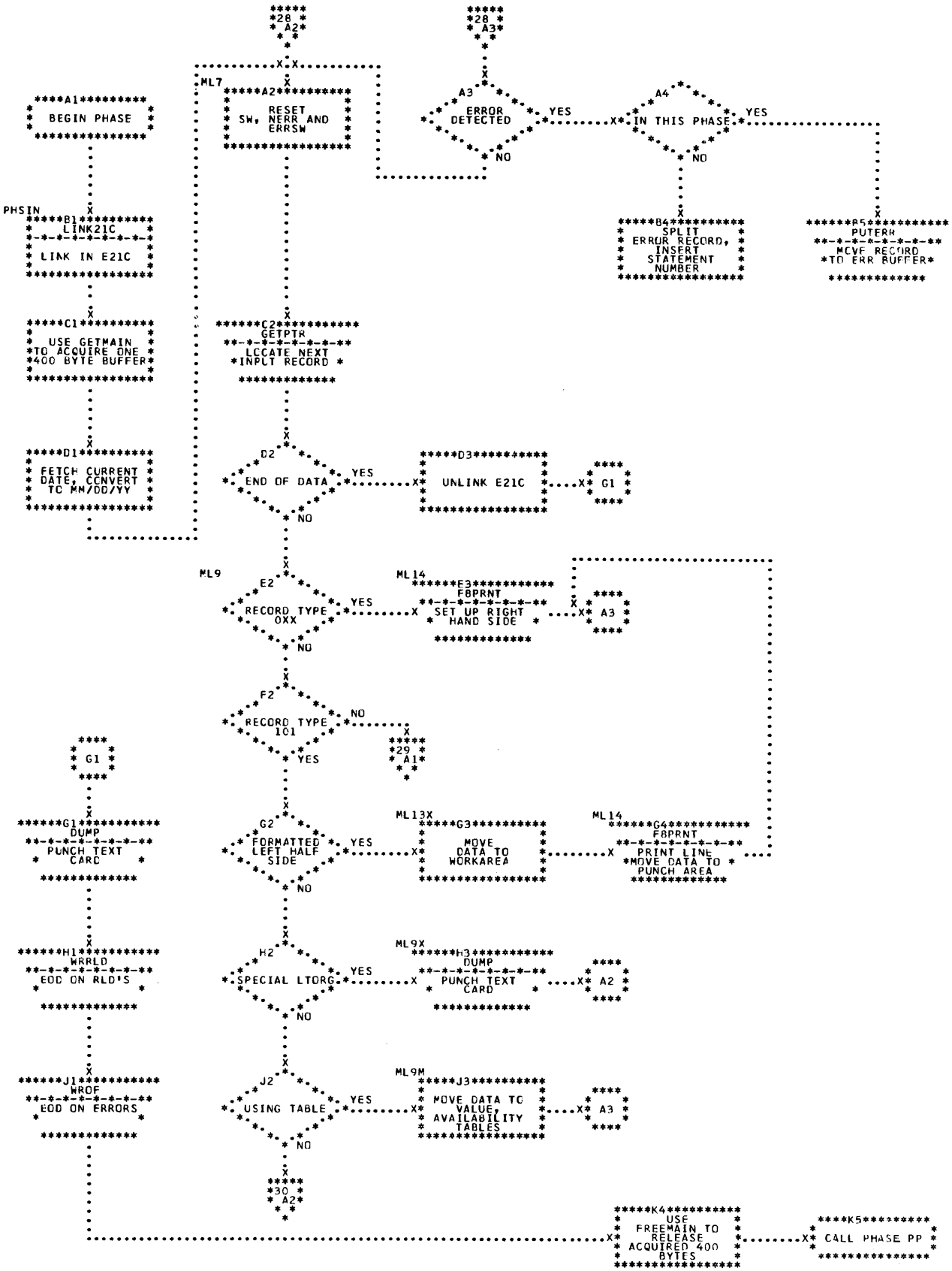


Chart 28. IET21B/21C/21D

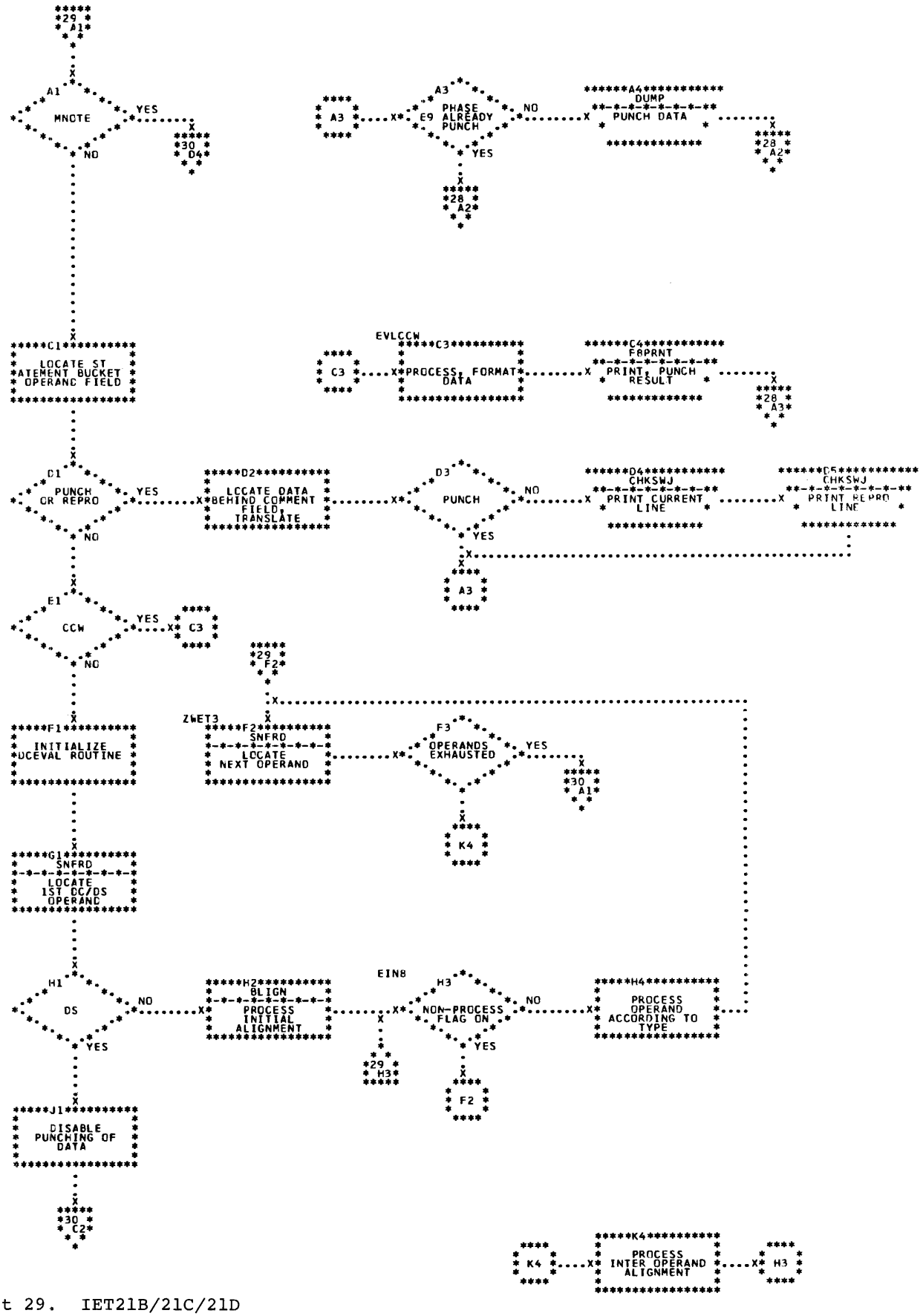


Chart 29. IET21B/21C/21D

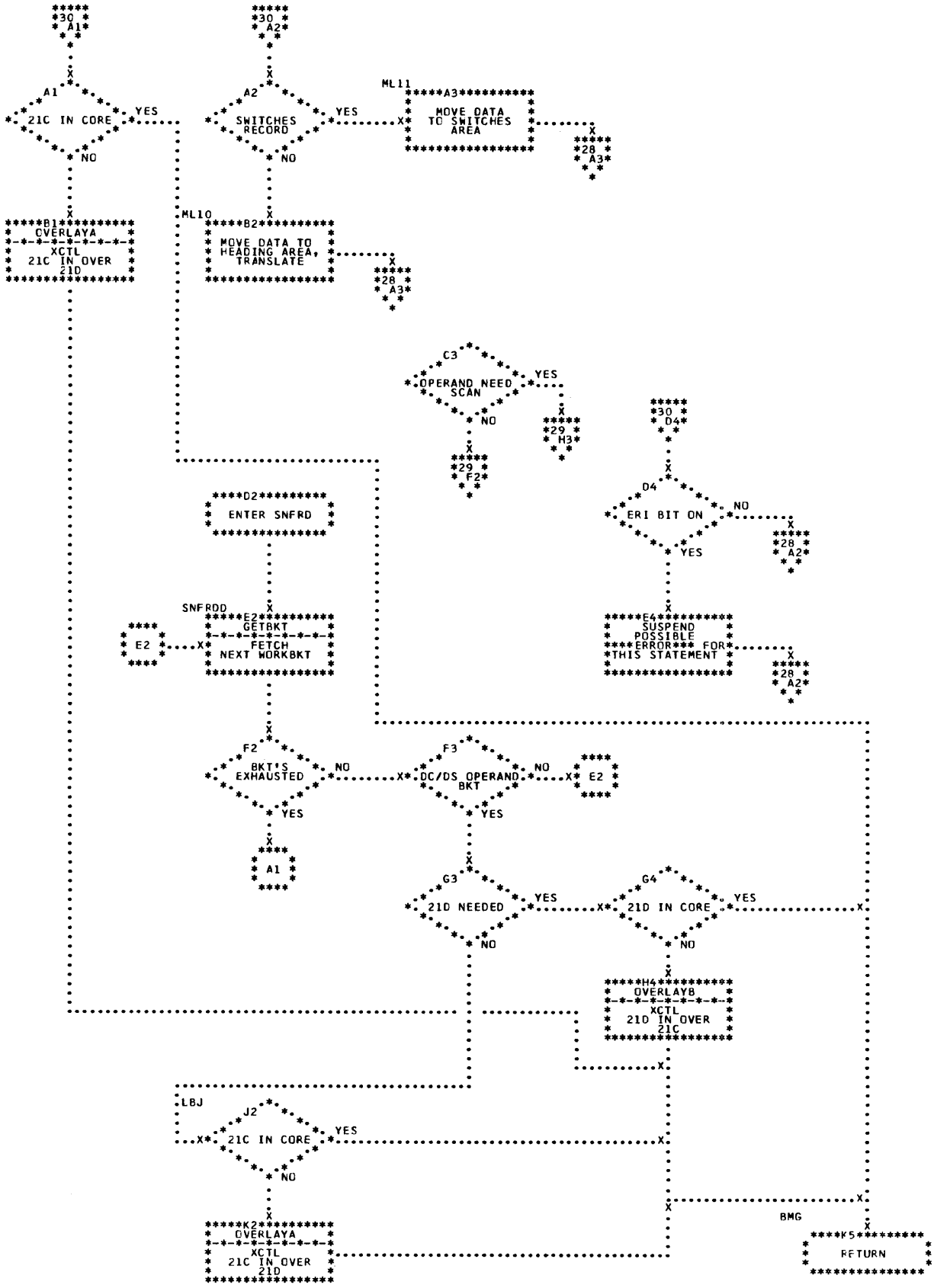


Chart 30. IET21B/21C/21D

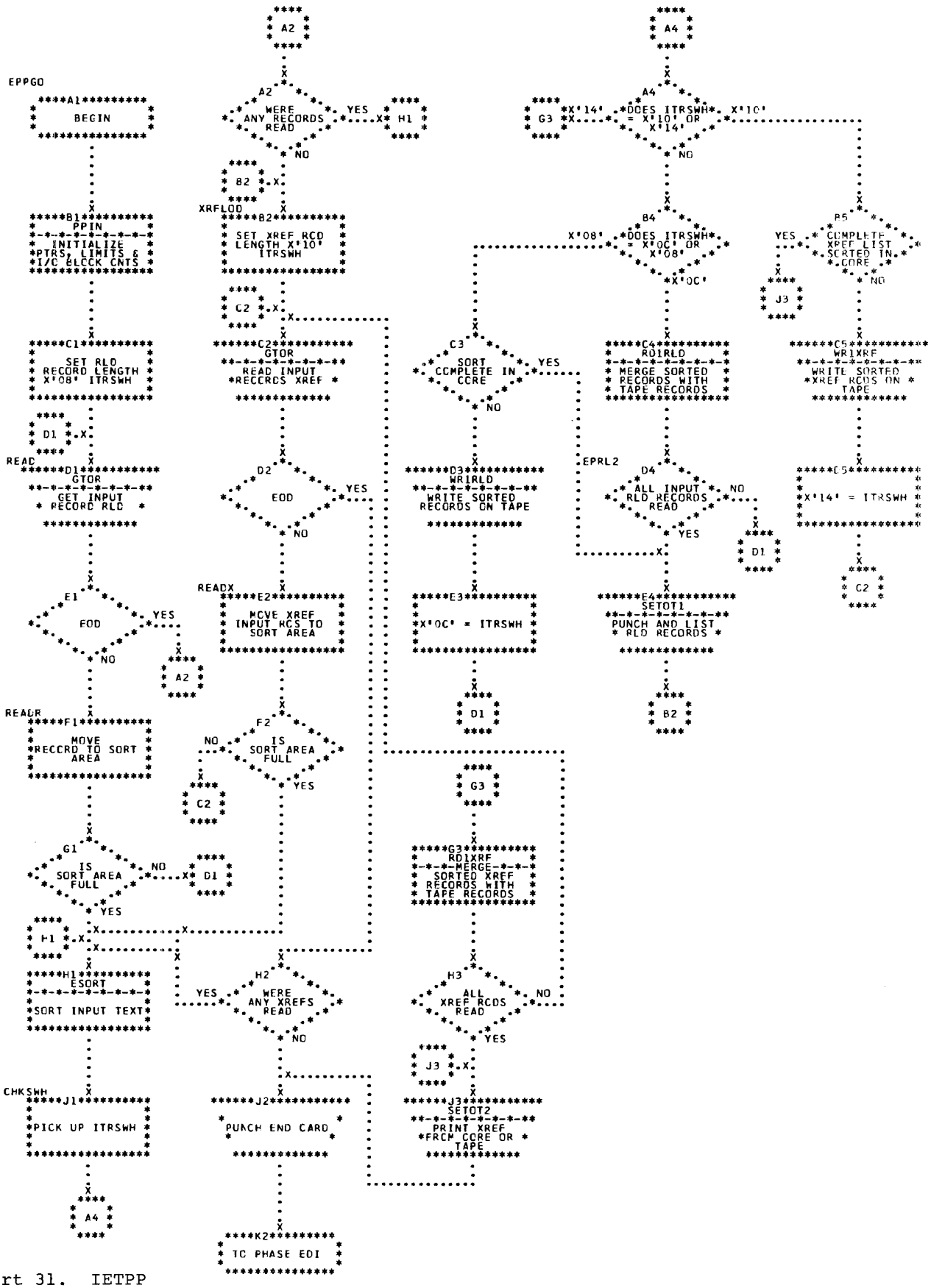


Chart 31. IETPP

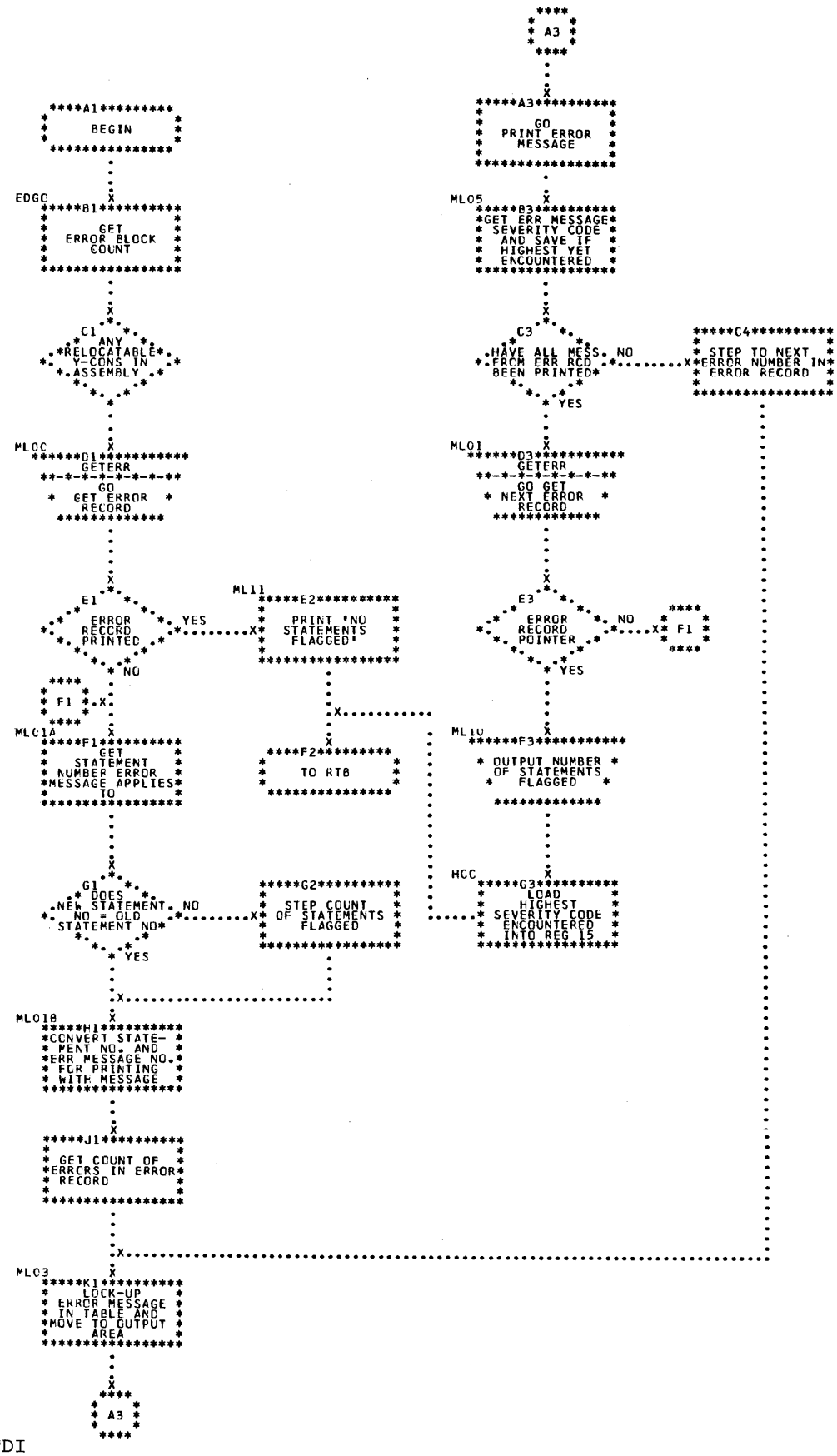


Chart 32. IETDI

APPENDIX A. INTERNAL ASSEMBLER INSTRUCTION CODES

Mnemonic	Decimal Value	Hexadecimal Value
GBLA	0	0
GBLB	1	1
GBLC	2	2
CLLA	3	3
LCLB	4	4
LCLC	5	5
SETA	6	6
SETB	7	7
SETC	8	8
AIF	9	9
AGO	10	A
ANULL	11	B
COPY	12	C
MACRO	13	D
MNOTE	14	E
MEXIT	15	F
MEND	16	10
ICTL	17	11
ISEQ	18	12
PRINT	19	13
SPACE	20	14
EJECT	21	15

Mnemonic	Decimal Value	Hexadecimal Value
PUNCH	22	16
REPRO	23	17
TITLE	24	18
ENTRY	25	19
EXTRN	26	1A
START	27	1B
CSECT	28	1C
DSECT	29	1D
COM	30	1E
EQU	31	1F
ORG	32	20
END	33	21
LTORG	34	22
USING	35	23
DROP	36	24
LITR	37	25
DC	38	26
DS	39	27
CCW	40	28
CNOP	41	29

APPENDIX B. TRANSLATE TABLE

All characters in source statements are translated to an internal hexadecimal coding (see below). Translation is done to facilitate comparisons and some arithmetic operations and to obtain a degree of character set independence.

The internal language is translated back to EBCDI code before output or in cases where the EBCDI code represents the actual binary value which is used as a mask, constant, etc.

The Translate Table also allows the user to compile programs written in other external card codes. A different translate table could be provided and the code translated to the same internal coding (or language).

The collating sequence of the internal language differs from the standard collating sequence in that numeric values are higher than alphabetic or special characters.

Standard Graphic Symbol	Machine Internal Hexadecimal Code	Standard Graphic Symbol	Machine Internal Hexadecimal Code
0	00	Q	1A
1	01	R	1B
2	02	S	1C
3	03	T	1D
4	04	U	1E
5	05	V	1F
6	06	W	20
7	07	X	21
8	08	Y	22
9	09	Z	23
A	0A	$	24
B	0B	#	25
C	0C	@	26
D	0D	+	27
E	0E	-	28
F	0F	*	29
G	10	/	2A
H	11	,	2B
I	12	=	2C
J	13	&	2D
K	14	.	2E
L	15	(	2F
M	16	)	30
N	17	'	31
O	18	blank	32
P	19		

DESCRIPTION OF RECORD	PHASE 2		PHASE 3		PHASE 4		PHASE 5	
	IN	OUT	IN	OUT	IN	OUT	IN	OUT
FIRST SOURCE RECORD OF A STATEMENT		01	01	15	15	08	08	08
MACRO INSTRUCTION LOGICAL STATEMENT RECORD		02	02					
PROTOTYPE STATEMENT CONTINUATION (LOGICAL STATEMENT RECORD)		03	03					
REPRO LOGICAL STATEMENT RECORD		04	04					
PROTOTYPE STATEMENT (FIRST) LOGICAL STATEMENT RECORD		06	06					
OPEN CODE SIGNAL RECORD (I.E. END OF MACRO DEFINITIONS)		07	07	17	17			
SOURCE STATEMENT CONTINUATION RECORD		08	08	08	08	08	08	
ERROR RECORD [WARNING MESSAGE]		08	08	08	08	08	08	
END OF FILE		0A	0A	0A	0A	0A	0A	
ERROR RECORD [NOT WARNING MESSAGE]		0D	0D	0D	0D	0D	0D	
ALL OTHER LOGICAL STATEMENT RECORDS		00	00					
C SECT, D SECT, START EDITED TEXT RECORD				01	01	01	01	
AGO EDITED TEXT RECORD				02	02	02	02	
AIF EDITED TEXT RECORD				03	03	03	03	
SETx & ACTR EDITED TEXT RECORD				04	04	04	04	
MACRO INSTRUCTION EDITED TEXT RECORD				05	05	05	05	
MACRO DEFINITION PROTOTYPE STATEMENT EDITED TEXT RECORD				06	06	06	06	
MEXIT EDITED TEXT FLAG RECORD				07	07	07	07	
ANOP EDITED TEXT FLAG RECORD				09	09	09	09	
MACRO INSTRUCTION OR PROTOTYPE OPERAND VALUE RECORD				0B	0B	0B	0B	
END OF MACRO INSTRUCTION OR PROTOTYPE RECORD				0C	0C	0C	0C	
STANDARD POINTER RECORD				10	10			
OPERAND LIST RECORD				11	11			
MEND EDITED TEXT FLAG RECORD				12	12	07	07	
MACRO PROTOTYPE HEADER POINTER RECORD				13	13			
MACRO PROTOTYPE POSITIONAL PARAMETER POINTER RECORD				14	14			
GLOBAL/LOCAL POINTER RECORD				16	16			
ALL OTHER EDITED TEXT RECORDS				00	00	00	00	

FLAGA

FLAGA bits apply to all SOURCE, LOGICAL STATEMENT, and EDITED TEXT records unless otherwise noted.

NOTE: For Type Indicators 07 and 0A, FLAGA will always be 00.

Bit	Mnemonic	Description
1	LRB	Last record on buffer indicator: 0 Not last record. 1 Last record.
2-4	RT	Record type: 000 Print as is. SOURCE record only. (These are assembly records created from program source input records in phase 2. SOURCE records can be constructed in phase 10B also. Construction in 10B will be from edited text for literals (ET-type 110) or MNOTE (ET-type 111). This record will step the statement number counter on a listing when not a continuation record.) 001 Error record (created in any phase). 010 Print as is but do not increment or display a statement number. SOURCE record only. (Created in phase 10B from edited text (type 110) for conditional assembly substituted statements outside of macro definitions.) 011 Print as is if GEN option is on; steps statement number counter. SOURCE record only. (Can be source record of comments for generation within macro definition (e.g., "*") in phase 2 or may be source record created in phase 10B from edited text (type 111) generated by macro instruction expansions.) 100 Process only. EDITED TEXT and LOGICAL STATEMENT records only. (Edited from source in phase 2.) 101 Do not construct SOURCE record or print. (In phase 7, dummy CSECT, ORG, and LTORG EDITED TEXT records for END statement processing. In phase 10B: Title EDITED TEXT record, listing control Switches record (created in association with PRINT, TITLE, SPACE, and EJECT) and Register Availability record, (Created in association with USING and DROP in phases 10B and 21A), and the 20-byte object record that replaces the edited text record (and is used to form the left-hand side of the print line.)

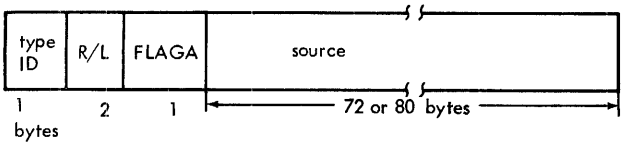
NOTE: When bits 2-4 are 101, bits 6-8 have one of the following meanings for the configurations shown:

Bit	Mnemonic	Description
6-8		010 Title record (created and substituted for TITLE ET in phase 10B). 011 Listing control Switches record (phase 10B). 100 Register Availibility Record (phase 10B). 111 Object record; used to form the left-hand side of the print line (phase 10B).
110		Process this record and construct SOURCE record for print (ET but no source, e.g., literals (phase 7) and conditional assembly substituted statements outside of macro definitions (phase 5)).
111		Process this record and construct SOURCE record for print if GEN option is on (Edited Text and MNOTE statements generated by macro expansions (phase 5)).

Bit	Mnemonic	Description
5	BF	Break flag: 0 No break. 1 Break (this logical record is continued in next buffer; a logical record can be broken only once).
6	ERI	Error record indication: 0 No error record follows. 1 Error record follows this record.
7	ESI	Equal sign indicator, EDITED TEXT records only (phase 7): 0 No equal sign in operand. 1 There is an equal sign in operand. NOTE: In phases 1 through 5, bit 7 has the following meaning: 0 SOURCE record is not continued. 1 SOURCE record is continued.
8	SLI	Symbol list iteration, EDITED TEXT records only (set and used by phase 8): 0 No. 1 Yes.
9-10	TO	Type of operation (bits 9 through 16 used as FLAGA only for EDITED TEXT and LOGICAL STATEMENT records): 01 Machine operation. 10 Assembler operation.
11	EMF	Extended mnemonic flag (used in phase 21B): 0 R1M is not meaningful. 1 R1M is meaningful.
12	BUG	Blow-up flag (can be set in phases 7 through 21A): 0 Record is processable. 1 Record is not processable.
13-16	R1M	M1 mask; the four-bit immediate field for extended mnemonic operation codes (used in phase 21B).

NOTE: Bits 9 through 16 are mainly set in phase 2. When in phase 7, they are for substituted mnemonic operation codes.

SOURCE RECORD



Type ID - See Appendix C.

R/L Record Length, a two-byte entry to indicate the length of the record.

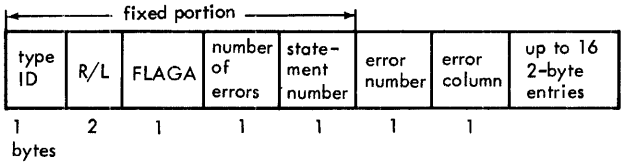
Source When created in phase 2 from SOURCE records from SYSIN or SYSLIB, source will be 80 bytes long.

When created in phase 10B from EDIT TEXT records, source will be 72 bytes long.

NOTE: If a SOURCE statement has a continuation card, an error, and is split between buffers (LRB, ERI, and ESI of FLAGA are all one), then the order on SYSUT1 is as follows:
source record (first part)
error record
source record (second part)
source continue record (from continuation card)

Source records are not created in phase 2 for the system macro definition source statements (with the exception of comments for generation ("*")).

ERROR RECORD



Type ID 08 - Warning message
0D - All other errors

R/L Record Length, a two-byte entry to indicate the record length which must be between 0007_{16} and $002B_{16}$

Number of errors The number of errors within a particular error record, from 01_{16} to 10_{16}.

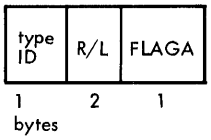
Statement number This byte is inserted in the record by phase 21B.

Error number Identification number of error or warning message.

Error column SOURCE statement column associated with previous ERROR NUMBER. If this column number is not appropriate, the value is 00_{16}.

NOTE: May be created in any phase up to phase 21B. There may be more than one partially filled record associated with a statement. There may be an error record associated with a source record which has no edited text record (e.g., a bad ICTL or ISEQ).

END OF DATA SET



Type ID $0A_{16}$

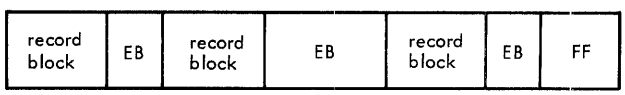
R/L Record Length, a two-byte entry to indicate the record length which must be 0004_{16}.

FLAGA 00_{16}

NOTE: Two of these records are inserted by phase 2 at the end of the text output stream. The records are dropped from the output stream by phase 7. In its place, at the beginning of the buffer, phase 7 attaches 4 bytes, 7F7F7F7F. This is the END OF DATA SET indicator for phase 8 through DI. After phase 7, the END OF DATA SET indicator, 7F7F7F7F, can appear anywhere in the buffer.

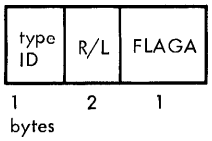
BLOCK FORMATS

Macro Generator Blocking
All macro generator utility segments are followed by an 'EB'. The last block is followed by an X'EB' and X'FF'.



Output Blocking (Assembler Input)
The macro generator output blocks are approximately 200 character blocks. Within the assembler flags with each record is an indicator which specifies the last assembler input block.

OPEN CODE SIGNAL



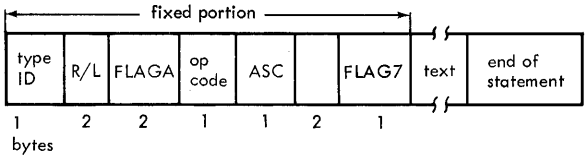
Type ID 07_{16} or 17_{16}

R/L Record Length, a two-byte entry to indicate the record length which must be 0004_{16}.

FLAGA 00_{16}

NOTE: This record is inserted after the last programmer macro definition by phase 2, if any are included. It is dropped from the phase 5 output stream.

LOGICAL STATEMENT



Type ID (see Appendix C.)

R/L Record Length, a two-byte entry to indicate the record length.

FLAGA (see Appendix C.)

Op code Hexadecimal Operation Code for machine instructions (see Appendix A for Assembler and Pseudo operation codes).

ASC Assembler Syntax Switches, inserted in phase 2 but not used by macro generator phase.

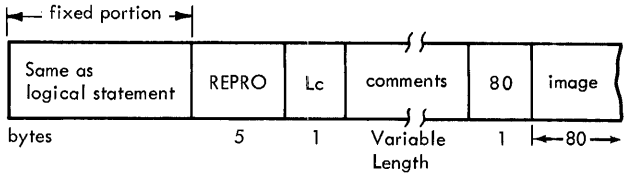
FLAG 7 01 - Macro Name in prototype statement is identical to a mnemonic operation code.
02 - Macro Name Field is in error in prototype statement.
04 - Macro prototype statement has a continuation record.

Text Relevant text from source record; all source between beginning and end column, and continue to end column, if continuation record. All blanks, except one, after end of comments field of last or only source record are dropped. As an exception, for macro prototype statements, one LOGICAL STATEMENT record is made for each source record. The following REPRO record is the other exception.

End of Statement FF_{16}

NOTE: LOGICAL STATEMENT records are not created for ICTL, ISEQ, MACRO, COPY, or COMMENTS statements (i.e., "*" or ".*"). The LOGICAL STATEMENT record is created in phase 2 and is used only to create a partially edited text record in phase 3.

REPRO



Lc Length of comments field, if any.

Image The 80 byte image of the source input record following the REPRO source record.

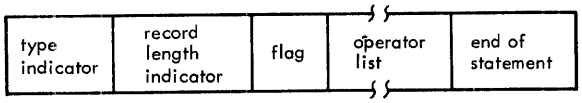
PARTIALLY EDITED TEXT

The Partially Edited Text records created in phase 3 are identical in format with the Fully Edited Text records produced in phase 4, except that the "a" (little a) pointer to the appropriate phase 5 dictionary is not inserted in phase 3 but space is reserved. The nine bytes of assembler flags shown in the fixed portion of the Logical Statement record remain the same in format and have the same functions as described.

DICTIONARY COLLECTION

The following records are used in the dictionary collection phase 4 to facilitate processing the stream of partially Edited Text records. These records are created in the Syntax Scan phase 3 in addition to the partially Edited Text records, but are discarded with only the Fully Edited Text records being passed to the Conditional Assembly phase 5.

Standard Pointer (associated with every ET record)

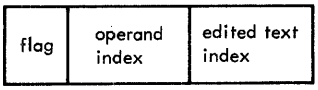


Type Indicator 10

Record Length Indicator XXXX, hexadecimal length of statement.

Flag, Bit 0 Not used
1 0, no operand record follows
1, operand record follows
2 0, no attributes provided
1, attributes provided
3 Not used
4 0, output Edited Text record to Phase 5 normally
1, suppress output of Edited Text record after processing in Phase 4.
5,6,7 (For GBL/LCL declaration only, otherwise not used.)
000, GBLA
001, GBLB
010, GBLC
011, LCLA
100, LCLB
101, LCLC

Standard Operator List (4 bytes for each entry in operand list record)



Flag If bit 1 = 0, format (a) is applicable;
If bit 1 = 1, format (b) is applicable.
(a) bit 0 1, dictionary action required
1 0, not a variable symbol
2 0, if normal symbol
1, if sequence symbol
3,4 00
5,6,7 actual length minus one of bytes at operand index (0-7)

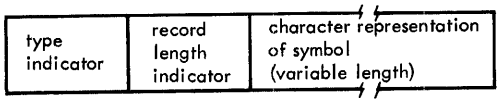
(b) bit 0 1, dictionary action required
1 1, variable symbol but not SYSLST
2,3 00, any variable symbol allowed
01, symbolic parameter required (SYSNDX and SYSLST not allowed)
10, must be SETB
4 1, subscript indicator
5,6,7 actual length minus one of bytes at operand index (0-7)

Operand Index (1 byte) XX, relative pointer to the associated symbol entry in Operand List record

Edited Text (2 bytes) XXXX, relative pointer to the position of the associated symbol text within the Edited Text record

End of Statement (1 byte) 00 (NOTE: If there is no OPERAND LIST, there will be no OPERATOR LIST)

Operand List



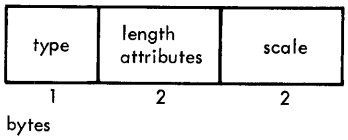
If the pointer record contains an operator list (OL), then the operand list record will follow the pointer record. The relative operand index pointer within the pointer record points to the associated Character Representation of Symbol found in the operand list record. This record is also created in the Syntax Scan phase 3 for use only in the Dictionary Collection phase 4.

Type Indicator 11

Record Length Indicator XXXX, hexadecimal value of operand list record length.

Character Representation of Symbol	Text (extracted from partially edited text record for every symbol or variable symbol in the associated edited text record that will require dictionary action (i.e., entry or look-up) in the Dictionary Collection phase 4). These (variable length) entries are pointed to by the pointer record.

If the Standard Pointer record for this Operand List record is preceded by an Edited Text record for a DC or DS, then the Character Representation of Symbol of the Operand List record will be preceded by five bytes of attributes.



Type See Appendix H.

Scale Fixed-point half-word representation.

GLOBAL/LOCAL POINTER

This pointer record has no Partially Edited Text record accompanying it. It is used to point to its associated Operand List record that follows and contains Global or Local set symbols, which are to be entered into the appropriate phase 4 dictionary. The record format and functions are identical to the Standard Pointer record, with the following exceptions:

Type Indicator (1 byte) X'16'

Edited Text Index field in the Operator List entry contains the set symbol dimension.

MACRO PROTOTYPE HEADER POINTER

This pointer record has no related Partially Edited Text record preceding it. This record is used to point to its associated Operand List record that follows and contains the macro mnemonic operation code and the variable parameter, if present, appearing in the name field of the associated prototype statement. This record serves mainly to initiate placement of the macro mnemonic in the Global dictionary and appraise the program that processing of a macro definition is beginning.

The record format and functions are identical to the Standard Pointer record, with the following exceptions:

Type Indicator (1 byte) X'13'

Edited Text Index field in the Operator List entry is not used for the entry associated with the macro mnemonic and contains a parameter sequence number of two if there is an entry for the name field.

MACRO PROTOTYPE POSITIONAL PARAMETER POINTER

This pointer record has no related Partially Edited Text record preceding it. This record is used to point to its associated Operand List record that follows and contains, in order, any positional symbolic parameters that appear in the Operand field of the associated prototype statement. These positional symbolic parameters will be entered in the appropriate Local dictionary in phase 4. This record and its associated Operand List record will not appear if there are no positional parameters present.

The record format and functions are identical to the Standard Pointer record, with the following exceptions:

Type Indicator (1 byte) X'14'

Edited Text Index field in the Operator List entry contains a sequence number that reflects the order of occurrence of the positional parameters in the prototype statement operand field.

Macro Prototype Statement Example:

	NAME	OPERATION	OPERAND FIELD
Statement:	&NAME	MACRO	&B, &C, &KEYWORD = 10

TYPE ID

13	macro header pointer		
11	MACRO	&NAME	Operand List Record

14	prototype positional pointer		
11	&B	&C	Operand List Record

08	source record	
06	macro prototype edited text	
10	standard pointer	

0B	KEYWORD=	10	Prototype Operand value Record
10	standard pointer		
11	&KEYWORD		Operand List Record

0C	number of keyword operands

FULLY EDITED TEXT

The Fully Edited Text is identical in format to the partially Edited Text. It becomes fully edited text when phase 4 inserts "a" pointers or symbolic parameter position numbers in the appropriate spaces reserved by phase 3 in the Partially Edited Text.

The "a" (little "a") pointers point to an appropriate entry in a phase 5 dictionary. These phase 5 dictionaries consist of the phase 4 dictionaries which were subsetted at the end of phase 4 and which phase 5 completes by inserting the results of Conditional Assembly evaluation. The Edited Text records can be divided into two classes. The first is machine instructions and all assembler instructions not included in the second class. The second class is edited text flag records (MEND, MEXIT) conditional assembly ET records (SETx statements, AIF, AGO), and macro instruction and macro prototype edited text.

CLASS I

Machine Instructions and all Assembler Instructions (except CLASS II)

type flag	R/L	assembler flags	name field	Y	operation field	Y	operand field	Y	comments field	Y	(E)

Type flag X'00' for all CLASS I statements except CSECT.
X'01' for all CSECT DSECT, START. [CSECT instructions are used in evaluating &SYSECT.]

R/L Same as previously described.

Assembler flags Same as previously described for LOGICAL STATEMENT records.

Name, operation, operand, and comments fields: (variable length)

Each field is terminated by an end of field flag (Y), X'F8'

All fields except the comments field may contain text that requires substitution for variable symbols, and/or evaluation.

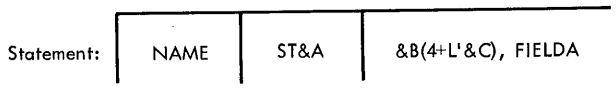
If the field requires no substitution or evaluation, the character string representing the field is preceded by a PUT flag and a byte indicating the length of the string.

PUT Example:

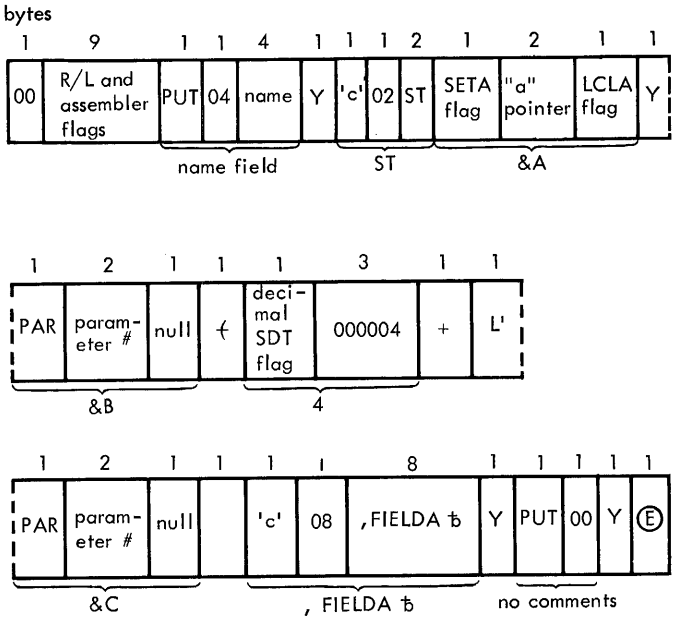
bytes

1	1		
PUT	06	symbol	Y
X'FD'			X'F8'

If a field contains variable symbols or text that requires evaluation, appropriate sub-records are inserted into the edited text in order to fully describe the field and the evaluation and/or the substitution action required. The format of each of the various types of subrecords is described in the evaluation routine formats. Following is an example of a model statement (macro definition) edited text record (as output by phase 3 for use in phases 4 and 5):



where: NAME is the name of the instruction, &A is a SETC symbol, &B is a macro prototype symbolic parameter corresponding to a macro instruction operand sublist, &B(4+L'&C) is a reference to an operand in the &B sublist, and &C is a macro prototype symbolic parameter.



'C' - Character string follows (next byte contains Length of String in bytes).

PAR - Operand request - request for macro instruction operand in place of symbolic parameter.

Decimal SDT Flag - Decimal Self Defining Term.

L' - Length attribute reference

Ⓔ - End of Statement. Appears at the end of all class one edited text records.

Macro Generator Output Record

The macro generator at the completion of the the Conditional Assembly (phase 5), passes to the assembler phases three record types.

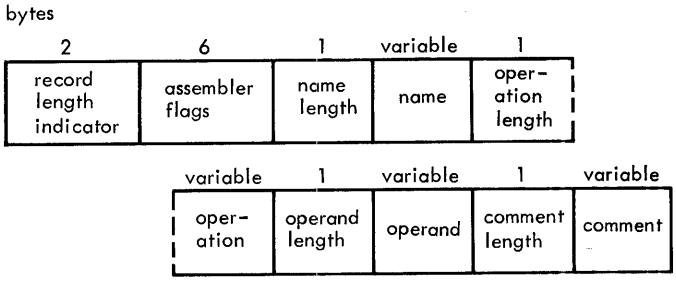
Source Record

The Source record contains source text which may be used to form the right-hand of the listing image output on SYSPRINT in phase 21B.

This record is the same as previously described, with the following exceptions:

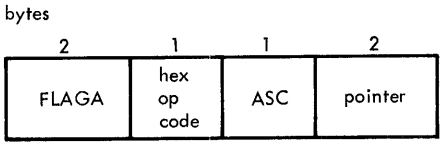
Type Indicator is removed.
Record Length Indicator is adjusted by one to compensate for the removal of Type Indicator.

Assembler Edited Text



Record Length Indicator, hexadecimal length of record.

Assembler Flags;



FLAGA the same as previously described.

hex op code and ASC, inserted in phase 7 for substituted mnemonic operation codes. All others had value inserted in phase 2.

pointer, phase 7 inserts the record length received from phase 5 in this space, thus maintaining the original record length value received from phase 5 and pointing to the first added on workbucket.

Name Length, length of name field. If this byte is zero, no name field will exist. Character Representation of Name (variable), statement name.

Operation Length, length of operation field. Character Representation of Operation (variable), statement operation.

Operand Length, length of operand field. If this byte is zero, no operand will exist. Character Representation of Operand (variable), statement operand.

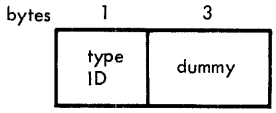
Comment Length, length of comments field. If this byte is zero, no comments will exist. Character Representation of Comments (variable), statement comments.

Error

The Error record is the same as previously described, with the following exceptions:
Type Indicator is removed.
Record Length Indicator is adjusted by one to compensate for the removal of the Type Indicator.

CLASS II

MEND



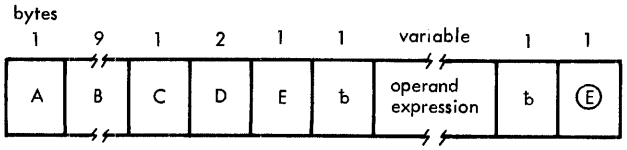
This record is created to signal the Dictionary Collection phase 4 to close out the macro being processed, subset its associated Local macro dictionary, and reinitialize the dictionary.
Type ID (1 byte), X'12' changed in output from phase 4 to X'07'.

MEXIT

This record has the same format as MEND.
Type ID (1 byte), X'07'

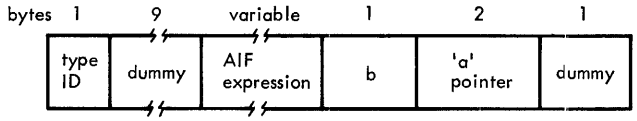
Conditional Assembly Edited Text

SET (SETA, SETB, and SETC)



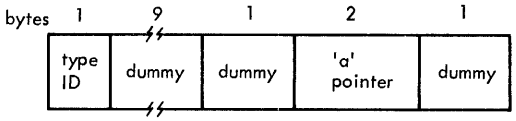
A - Set Statement flag (X'04')
B - R/L and Assembler Flags
C - Set variable flag (SETA = X'28', SETB = X'29', SETC = X'2A')
D - 'a' pointer
E - Global or Local flag (When the Set variable is subscripted, bits 4-7 indicate the subscript number.) Global = '80' or Local = '00'.
b - Blank (X'32')
E - End of statement

AIF



The operand field of Set Statements and AIF Statements will contain expressions requiring substitution and/or evaluation. These expressions are fully described in the ET by appropriate sub-records. The format of each of the various types of sub-records is described in the evaluation routine formats. The way in which these sub-records are used to describe an expression is shown in Class II Section.

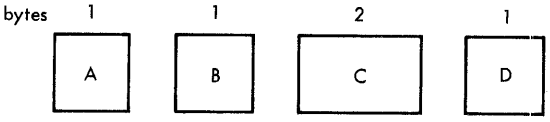
AGO



EVALUATION ROUTINE

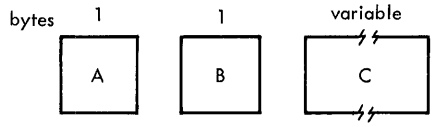
These sub-records are used to describe expressions that require substitution and/or evaluation.

Attributes (L'I'S'T')



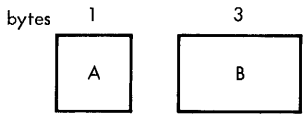
A - Flag byte (type of attribute) see separate list
B - Symbolic parameter (X'39') or symbol flag (X'FA') see separate list
C - 2-byte "a" pointer, if reference is to an attribute of a symbol, or parameter position number in low order byte of C if the reference is to an attribute of a symbolic parameter
D - Dummy

CHARACTER STRING



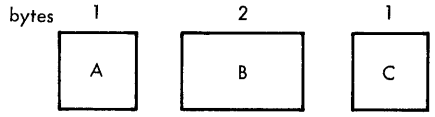
A - Flag byte 'C' = X'27'
B - True length byte (if zero, C will not exist)
C - Data bytes (variable bytes of characters)

DECIMAL, HEX, BINARY, OR CHARACTER SELF-DEFINING TERM



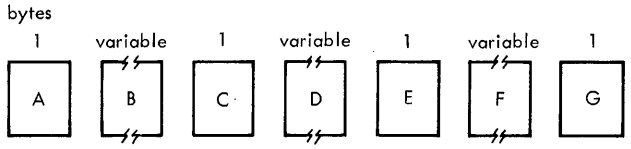
A - Flag byte (Hexadecimal = X'22', Binary = X'23', Decimal = X'24', Character = X'25')
B - Data bytes, three bytes of data in binary

VARIABLE SYMBOL



A - Flag byte (SETA = X'28', SETB = X'29', SETC = X'2A')
B - 2-byte "a" pointer
C - Bits 5-7 for subscripted SETB, indicate in binary form, the particular bit (0-7) within the byte referenced by B that contains the SETB evaluation, bit 0 = 0 for Local or 1 for Global

SUBSTRING



A - Begin substring 'BEGSUB' flag (X'2C')
B - Character expression (variable bytes)
C - Begin first operand 'SUBOPE' flag (X'2D')
D - Expression 1 (variable bytes)
E - First operand completed 'SUBCOM' flag (X'2E')
F - Expression 2 (variable bytes)
G - End of substring notation 'SUBCLS' flag (X'2F')

SUBSCRIPTING

Left parenthesis is replaced by a special 'SUBSCRIPT LEFT PAREN FLAG' (X'03')
Remainder of the format is as previously described.

CONCATENATION

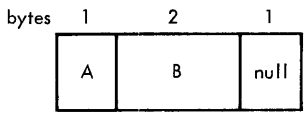
Occurs automatically by just eliminating the period. Two character strings (or Set variables), one immediately following the other, will be concatenated, and no concatenation flag required.

OPERAND REFERENCE

Reference in a macro definition model statement or inner macro instruction to a (prototype operand) symbolic parameter is made by position.
Operands are numbered as follows:

0 $SYSNDX
1 $SYSECT
2 Symbolic parameter in name field of prototype statement
3 ↕ 203 } Operand field of prototype statement

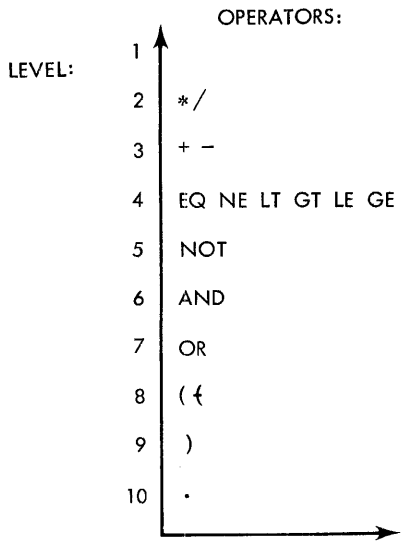
Keyword operands are given a position number similar to positional operands. The positions are assigned in the order that they appear in the operand field of the prototype statement.
Format of request for substitution of macro instruction operand in place of symbolic parameter:



A - Operand request flag PAR (X'39')
B - Operand number

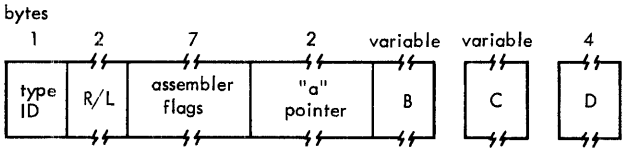
EVALUATION ROUTINE OPERATOR HIERARCHY

The hierarchy of operations performed in evaluating expressions in phase 5 is schematically illustrated by the following.



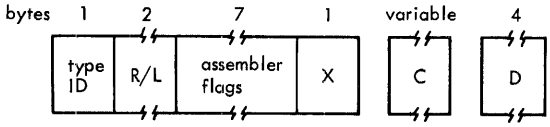
MACRO INSTRUCTION

General



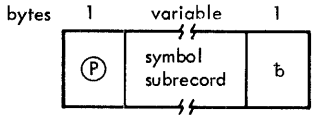
PROTOTYPE STATEMENT

General



Type ID — Macro Instruction '05'
Prototype Statement '06'

B — If a symbol appears in the name field of a macro instruction, a sub-record will appear in the following format:



Ⓟ — Positional macro instruction operand follows (X'FB')

Symbol Subrecord — A sub-record describing the symbol in the (macro instruction) operand value format

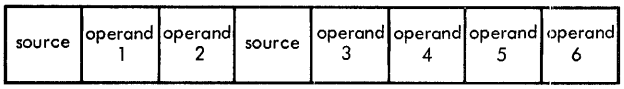
b — Blank (X'32')

X — Only keyword operands are described in the macro prototype statement edited text record. This field contains a count of the number of positional operands in the prototype statement operand.

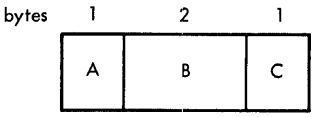
C — A variable number of operand value edited text records. In a macro instruction, there is one operand value edited text record for each macro instruction operand. In a macro prototype statement, there is one operand value edited text record for each keyword variable parameter appearing in the operand field of the prototype statement.

There can be source or error records interspersed between any operand value record.

Example:

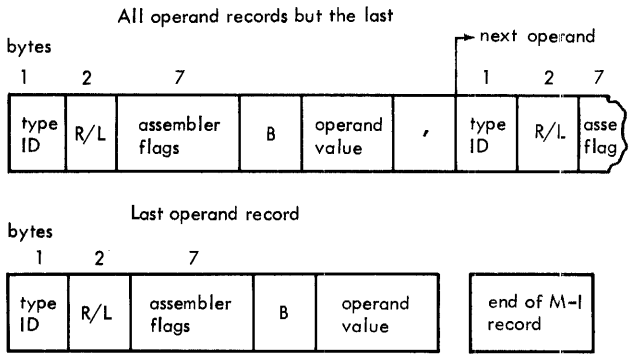


D — End of macro instruction or prototype statement edited text record.



A - End of edited text flag (X'0C')
B - Record length 4 bytes
C - Number of keyword operands in instruction operand

OPERAND VALUE RECORDS



Type ID — Operand record flag (X'0B')

B — Position or keyword flag

Ⓟ Positional (X'FB')

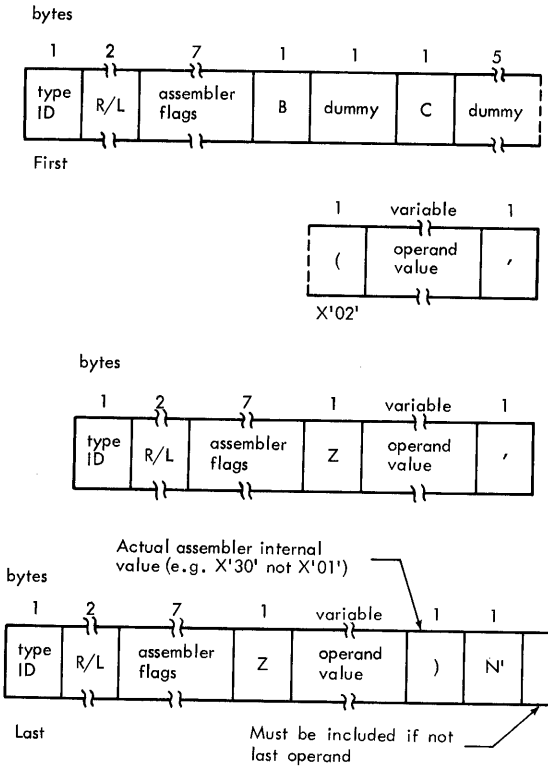
Ⓚ Keyword (X'FC')

Operand Value	Macro instruction and prototype operands are described by appropriate sub-records. The format of the various sub-records are described under Operand Value Formats.

EOB - End of Block

A macro instruction or prototype may be continued in the next block. An 'EOB' flag (X'FE') will appear where 'OPERAND RECORD FLAG' normally appears.

SUBLIST OPERANDS



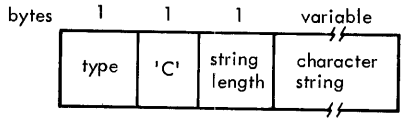
NOTE: Source records may appear between Sublist operands

Type ID	Operand record flag (X'0B')
B	This is the positional or keyword flag which appears only this time for the entire sublist
	Ⓟ Positional (X'FB')
	Ⓚ Keyword (X'FC')
C	Sublist flag (X'F0')
Z	Continue Sublist flag (X'F9')
Operand Value	Described in operand value format.
N'	The number of operands in the Sublist.

OPERAND VALUE FORMATS

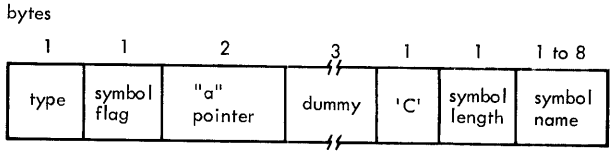
The following formats are used to describe macro instruction and macro prototype operands. Each operand value record must be preceded by a Ⓟ or Ⓚ flag except for macro instruction sublist operands in which only the first sublist operand is preceded by a Ⓟ or Ⓚ flag.

Character String



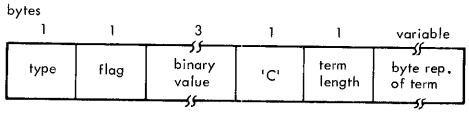
Type	See type byte table
'C' Character String Flag	X'27'
String Length	Length of character string in bytes
Character String	

Symbol



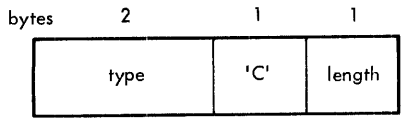
Type	See type byte table
Symbol Flag	FA
"a' Pointer	Dictionary location of symbol record
Dummy	Not used
'C' Character String Flag	X'27'
Symbol Length	Length of symbol name in bytes
Symbol Name	

Decimal, Character, Hexadecimal, or Binary Self-Defining Term



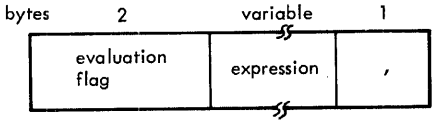
Type	See type byte table
Flag	'22' for hexadecimal self-defining '23' for binary self-defining '24' for decimal self-defining '25' for character self-defining
Binary Value	Binary value of term
'C' Character String Flag	X'27'
Term Length	Length of term in bytes
Byte Representation of Term	

Omitted Operand



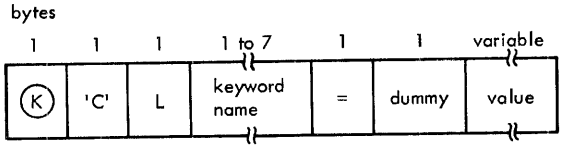
Type See type byte table
'C' Character String Flag X'27'
Length 00

Operands to be Evaluated



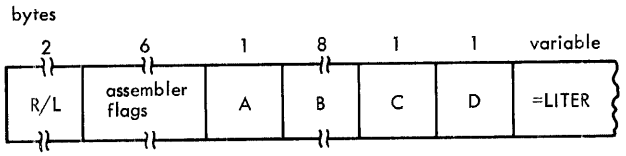
Evaluation Flag X'26'
Expression In evaluation routine format

Keyword



(K) Keyword X'FC'
'C' Character String Flag X'27'
L Length of keyword name + one
Keyword Name
= X'2C'
Dummy
Value The keyword value is described by operand value formats. Value of a keyword may also be a Sublist.

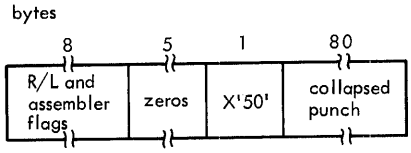
ASSEMBLER EDITED TEXT FOR LITERAL



R/L - Record Length, same as previously described.
Assembler Flags - Same as previously described.
A - Name Length (X'08'). Name length will be '00' when there is no reference to the Location Counter (*) in the literal.
B - Statement Name Field, contains statement number when A is equal to '08'. The statement name field is dropped from the record when A is equal to '00'.
C - Operation Length, always '00' for this type record.
D - Operand Length, hexadecimal representation of operand length. LITERAL, variable length XXrecord containing the literal in character format.

PUNCH ASSEMBLER EDITED TEXT (94 bytes)

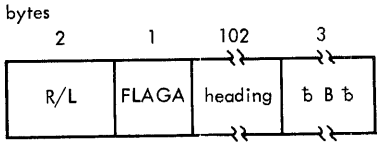
Reformatted version of normal Assembler Edited Text Records, accomplished in phase 10B.



R/L - Record Length X'5E'
Collapsed Punch - means every double apostrophe (") and every double ampersand (&&) is reduced to a single apostrophe or ampersand.

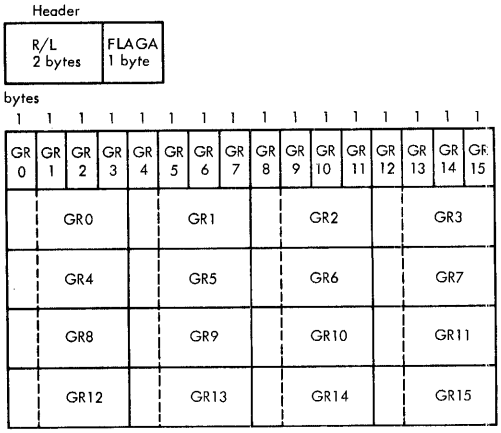
TITLE ASSEMBLER EDITED TEXT (108 bytes)

Reformatted from normal Assembler Edited Text Records in phase 10B.



R/L - Record Length X'6C'
Heading - The information in the operand field of the title statement, including beginning and ending quote marks.
B - A three byte field containing a blank, followed by a single apostrophe, and followed by another blank to ensure termination of the scan.

REGISTER AVAILABILITY TABLE/RECORD (83 bytes including 3-byte header)



The Register Availability Table/Record consists of two parts: a 16-byte Availability Table, and a 16 times 4-byte Value Table.

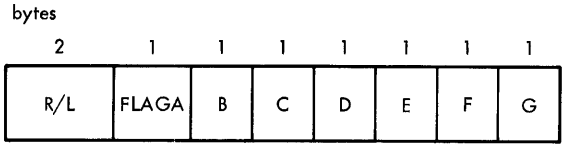
The 16 bytes of the Availability Table are numbered 0 through 15 to correspond to General Register (GR) 0 through 15: Byte i = 0 indicates that General Register i (GRi) is not now in use as specified by Assemble USING and DROP statements, and Byte i ≠ 0 indicates that General Register i (GRi) is in use.

The first four bytes of the Value Table correspond to General Register 0 (GR0), the second four bytes to GR1, etc. The four bytes corresponding to a General Register have the following meaning when the General Register is in use:

Byte 1 ESD-ID (For calculation of S-type Adcons.)
Bytes 2-4 Location Counter Value as specified in the pertinent USING statement.

R/L - Record Length X'53'
FLAGA - See Type Indicator and FLAGA appendix.

SWITCHES TABLE/RECORD



R/L - Record Length X'09'
FLAGA - See Type Indicator and FLAGA appendix.
Flags B through D - the current status of the listing options:

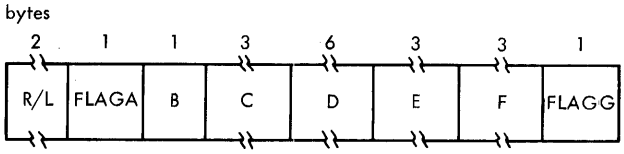
B	'00'	ON
	'FF'	OFF
C	'00'	GEN
	'FF'	NOGEN
D	'00'	DATA
	'FF'	NODATA

Flags E and F - indicates whether the assembler should take forms control action corresponding to the assembler SPACE and EJECT instructions:

E	'00'	Associated instruction is not SPACE
	'FF'	Associated instruction is SPACE
F	'00'	Associated instruction is not EJECT
	'FF'	Associated instruction is EJECT

Flag G - The value in the operand field of SPACE statement, expressed in binary form.

OBJECT RECORD (20 bytes)



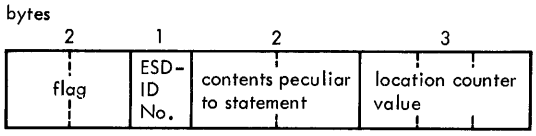
R/L - Record Length X'14'
FLAGA - See Type Indicator and FLAGA appendix.
B - ESD/ID Field. Normally indicates the current ESD identification number. When X'00', indicates that this statement should not be punched (i.e., statements in COM or DSECT control sections).
C - Location Counter Field. The value that will appear in the Location Counter Field on a listing. Normally this is the appropriate value of the Location Counter. However, for some statements (e.g., USING, END, EQU), this represents the value of the first or only expression in the operand field of the statement.
D - Machine Instruction Field. Normally this field contains a machine instruction in object format. As a result of the CNOP instruction, this field consists of zero to three half-word fields, containing X'0700'. If the object record is being used for alignment, it will contain six bytes of zero.
E - Effective Address Field Number 1. Under normal conditions, this field will contain the value that will appear in the ADRR1 field on a listing. This is the effective address of the first operand of a statement, if applicable, or blank. If the object record is being used for alignment, it will contain zero.
F - Effective Address Field Number 2. This field usually contains the value that will appear in the ADRR2 field on a listing. This is the effective address of the second operand of a statement, if applicable, or blank. If the object record is being used for alignment, the low-order byte will contain a number from 1-7 in binary form, indicating the number of bytes of zeros to be used from fields D and E for alignment.

Flag G -

Bit		Description
1-4	0000	Will appear only if bits 5-8 are all ones.
	0001	Not used.
	0010	Use the first half-word of field D for output (e.g., RR instructions, one CNOP '0700' half-word).
	0011	Not used.
	0100	Use the first two half-words of field D for output (e.g., RX, RS, and SI instructions

Bit		Description
		or two CNOP '0700' half-words).
	0101	Not used.
	0110	Use the first three half-words of field D for output (e.g., SS and three CNOP '0700' half-words).
	0111	Not used.
	1000	Not used.
	1001	Not used.
	1010	Not used.
	1011	Not used.
	1100	Not used.
	1101	Not used.
	1110	Not used.
	1111	Will appear only if bits 5-8 are all zeros.
5-8	0000	Do not print Location Counter (Field C) (e.g., DROP statement, bad END statement, etc.).
	0001	Not used.
	0010	Only field E should appear on a listing.
	0011	Both fields E and F should appear on a listing.
	0100	Indicates that this object record is a DC (this object record appears in main storage only during phases 21B, 21C, and 21D).
	0101	Not used.
	0111	Not used.
	1000	Indicates that this object record is being used for alignment (e.g., 1-7 bytes of zeros or 0-3 CNOP '0700' half-words).
	1001	Not used.
	1010	Not used.
	1011	Not used.
	1100	Not used.
	1101	Not used.
	1110	Not used.
	1111	If the punch buffer is partially full or full, output on SYSPUNCH. Then use a complete SYSPUNCH record for this statement (e.g., DS, ORG, and LTORG statements).

STATEMENT WORKBUCKET (8 bytes)



Flag

Bit	Set in Phase	Description
1-3	7	WBT- Workbucket type 001 Indicates Statement Workbucket
4-6	7	Alignment, number of bytes to be inserted before this statement to align the statement on proper boundaries.
7	7	DC/DS Switch 0 DC/DS 1 Other
8	7	*Location Counter reference in literal 0 None (no literal) 1 Yes
9	8	ADJCA- Adjective Code A. This code is meaningful only if this statement has a name. 0 Name not defined within DSECT or COM 1 Name is defined within DSECT or COM
10-11	8	ADJCB- Adjective Code B. This code is meaningful only if this statement has a name or is an EXTRN. 00 This statement is a CSECT or START 01 This statement is an EXTRN 10 This statement is a DSECT 11 This statement is none of the above
12	8 and/or 10	Location Counter wraparound. 0 No LC wraparound 1 LC wraparound
13	8	ESDP- ESD Processed Flag. 0 Unprocessed 1 Processed
14	10	Statement bucket adjustment. 0 No statement bucket Location Counter value has been adjusted by E10 1 Statement bucket Location Counter value has been adjusted by E10
15-16	8	NSL- Name in Symbol List Flag. Will be used in ESD processing. 00 Statement name not found in Symbol List 01 Not used 10 Name found and no error detected 11 Name found and error detected
ESD-ID NO.		The External Symbol Dictionary identification number set in Phase 8

Contents Peculiar to Statement

For EQU's, the content is the length of the first term (flag byte is not changed). For CSECT, START, COM, DSECT, ENTRY, or EXTRN, the low-order 4 bits of the two-byte field signify the type of statement as follows:

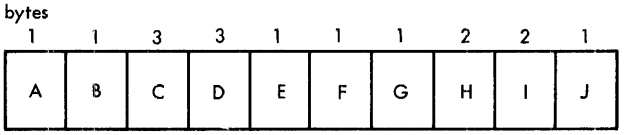
Hexadecimal Digit	Meaning
'0'	Named CSECT or Named START
'1'	ENTRY
'2'	EXTRN
'4'	Unnamed CSECT or Unnamed START
'5'	COM
'7'	DSECT

For DC/DS, the content is the length attribute of the first constant of the first operand. For CNOP, the low-order hexadecimal digit contains the number of bytes to be generated to force proper alignment. For Machine Instructions, the content is the length attribute of

the name, if any. For REPRO and PUNCH, low-order byte is 'FF' if the card is punched before the ESD cards. Information is used by phase 21B.

Location Counter Value — The current value of the Location Counter. Set in Phase 8, adjusted in Phase 10.

DC/DS OPERAND WORKBUCKET (16 bytes)



A - Flag

Bit	Set in Phase		Description
1-3	7	WBT-010	Workbucket type Indicates DC/DS Operand Workbucket
4	7	LMP-	Length Modifier Present
		0	No. If this is a P or Z type constant and this operand field contains multiple constants, additional constants may have different lengths than the first constant. LMP = 0 only when LMM = 0.
		1	Yes.
5	7	LMM-	Length Mode Modifier
		0	Length field contains total number of bytes for this operand
		1	Length field contains total number of bits for this operand
			NOTE: This is used in determining the increment of the Location Counter from operand to operand in a DC/DS Statement.
6	8		Reference to Location Counter
		0	No workbucket contains reference to * (Location Counter)
		1	At least one workbucket contains reference to *
7	7		Not used
8	7 or 8		Processing indicator
		0	Continue processing
		1	Process no further

B - Type of Constant Translated

Printer Graphic Original Form	Hexadecimal Translated Form
C	'00'
X	'01'
B	'02'
P	'03'
Z	'04'
D	'05'
E	'06'
F	'07'
H	'08'
A	'09'
Y	'0A'
V	'0B'
S	'0C'
Any other alphabetic character	'FF'

C - Length Field, meaning of the constants depends upon LMM switch.

D - Duplication Factor, set in phase 7 or 8. Left blank if an expression, phase 8 evaluates in this case.

E - Number of Constants in Operand, binary notation. Set in phase 7.

F - Pointer, pointer to quote or left parenthesis preceding the first operand constant in the edited text portion of the record. Placed in phase 7.

G - Exponent Modifier

Bit	Set in Phase		Description
1	7 or 8		Sign indicator
		0	Plus
		1	Minus
2-8	7 or 8		Base 10 exponent modifier
			NOTE: Left blank if an expression, phase 8 evaluates in this case.

H - Flag

Bit	Set in Phase		Description
1	7 or 8		Scale modifier sign
		0	Plus
		1	Minus
2-11	7 or 8		Scale modifier value

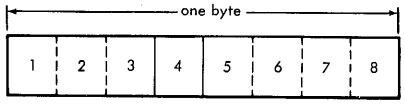
Bit	Set in Phase	Description
		NOTE: Left blank if an expression, phase 8 evaluates in this case.
12	7	VWB- Variable Workbucket Flag 0 No variable workbuckets follow 1 Variable workbuckets follow
13-15	8	Alignment, number of bytes to insert before this operand.
16	8	TNT 0 DS with constant 1 DS with type but no constant

I - Length Modifier Value, constants depend upon LMP switch. When LMP = 0, contains (explicit or implicit) length of the first constant of the first operand. When LMP = 1, contains length modifier value per constant (bits if LMM = 1, bytes if LMM = 0). Left blank if an expression, phase 8 evaluates in this case.

J - Flag, meaningful if VWB = 1

Bit	Set in Phase	Description
1	7	Expression for duplication factor 0 No 1 Yes
2	7	Expression for length expression 0 No 1 Yes
3	7	Expression for scale modifier 0 No 1 Yes
4	7	Expression for exponent modifier 0 No 1 Yes
5		Not used
6-8	7	Number of types of modifier expressions present

OPERATOR-DELIMITER WORKBUCKET (1 byte)



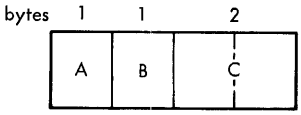
Bit	Set in Phase	Description
1-3	7	Workbucket type 000 Indicates Operator-Delimiter Workbucket
4		Not used
5-8	7	Workbucket identity 0000 + (plus) 0001 - (minus) 0010 * (asterisk) 0011 / (slash) 0100 , (comma) 1000 ((left parenthesis) 1001) (right parenthesis) 1011 b (blank)

NOTE: Parenthesis used sytactically to enclose duplication factor, length, scale, or exponent expressions, or to enclose adcon(s) are not marked by an Operator-Delimiter Workbucket.

The comma workbucket is used wherever a comma is found in a machine instruction operand and when it separates symbolic constants in an adcon. It is also used as a separator between duplication factor, length, scale, and exponent expressions when present.

The blank workbucket is used as a delimiter for the operand (there may be multiple operands).

SYMBOLIC LENGTH WORKBUCKET (4 bytes)



A - Flag

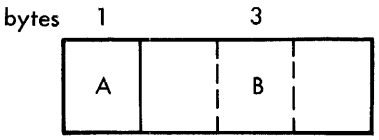
Bit	Set in Phase		Description
1-3	7	WBT-	Workbucket type 100 Indicates Symbolic Length Workbucket
4		LSV-	Length of Symbol/Value Switch
	7	0	Low-order byte of value field contains length in bytes less one, of the symbol whose length attribute is desired in the text.

Bit	Set in Phase		Description
	8 or 10	1	Value field contains length attribute, (true length minus one) of the symbol.
5	8 or 10	PHS- 0	Phase Switch Length attribute needed by phase 8 in connection with expressions including symbols requiring previous definition.
		1	Length attribute needed by phase 10, but not by phase 8.
6	10	XREF- 0	Cross Reference Switch Cross reference has not yet been made
		1	Cross reference has been made by phase 10
7-8		Not used	

B - Pointer, pointer to the symbol whose length attribute is desired. The pointer is relative to the first byte of the operand field in the assembler edited text record and points to the left-most byte of the symbol.

C - Value, Contents are dependent upon the LSV switch.

SELF-DEFINING TERM WORKBUCKET (4 bytes)

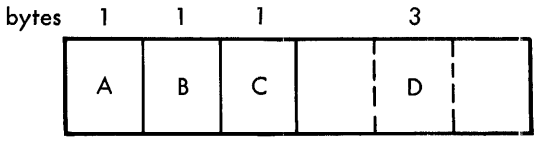


A - Flag

Bit	Set in Phase		Description
1-3	7	WBT- 101	Workbucket type Indicates Self-Defining Term Workbucket
4-6		Not used	
7-8	7	Class 00 01 10 11	 Decimal Hexadecimal Binary Character

B - Value, contains the true 24-bit value. For the C (Character) Type, these three bytes are in EBCDIC, not in internal translated code. This workbucket is processed in phase 7.

OPERAND FIELD SYMBOL WORKBUCKET (6 bytes)



A - Flag

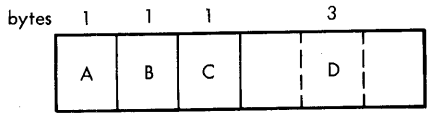
Bit	Set in Phase		Description
1-3	7	WBT- 110	Workbucket type Indicates Operand Field Symbol Workbucket
4		LSV-	Length of Symbol/ Value Switch
	7	0	Low-order byte of value field contains length in bytes less one of the symbol whose ESD-ID and/or Location Counter value is desired
	8 or 10	1	Value field contains Location Counter value of the symbol. This value is un-adjusted if PHS = 0, and is adjusted if PHS = 1
6	10	XREF- 0	Cross Reference Switch Cross reference has not yet been made. LAP field contains pointer to left-most byte of the symbol in the assembler edited text record.
		1	Cross reference has been made by phase 10. LAP field contains length attribute of symbol.
7-8	8 or 10	ELS-	Excess Length Switch
		00	LAP field contains length attribute of symbol (true length minus one)
		10	LAP field is too small to contain length attribute (true length is greater than 256)

B - ESD-ID, set in phase 8 or 10, contains ESD identification number. An absolute term (evaluated in phase 8) is given an ESD-ID number of zero.

C - LAP, Length Attribute Pointer, contents depend upon XREF switch.

D - Value, contents depend upon LSV switch.

LITERAL WORKBUCKET (6 bytes)

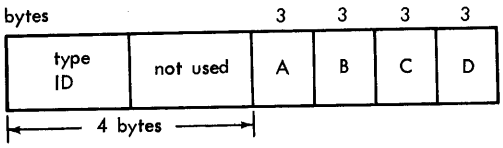


A - Flag

Bit	Set in Phase		Description
1-3	7	WBT-111	Workbucket type Indicates Literal Workbucket
4		LPE-	Literal Pool/ESD-ID Switch
	8	0	ESD-ID field contains literal pool number, value field contains displacement relative to literal pool number and string
			NOTE: Each LTORG or END produces a literal pool. These each have a number (literal pool number); however, the number is not currently used because the order of the literal table segments on SYSUT1 implies the information.
	10	1	ESD-ID field contains ESD-ID number, value field contains Location Counter.
5-6			Not used
7-8	7		Literal String Numbers 00 1-byte string 01 2-byte string 10 4-byte string 11 8-byte string

B - ESD-ID Field, contents depend upon LPE switch.
C - Set aside for length attribute of literal, filled in phase 10.
D - Value Field, contents depend upon LPE switch.

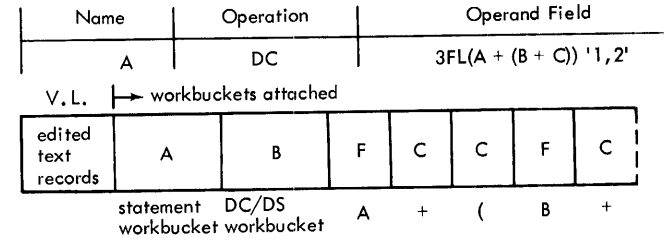
LTORG STATEMENT WORKBUCKET (16 bytes)

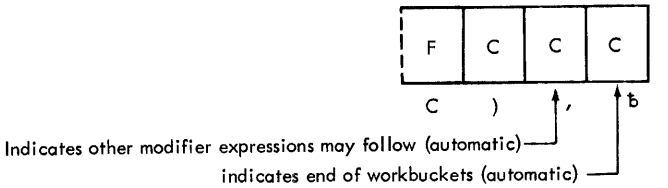


Type ID — Workbucket type
010 — Indicates LTORG Statement Workbucket

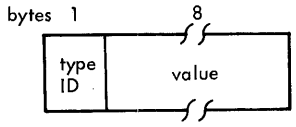
A - Total Displacement of 8-byte Chain, this field indicates the sum of the object lengths of all literals in the 8-byte string in the associated literal pool. Filled in Phase 7.
B - Total Displacement of 4-byte Chain, same as A, except for 4-byte string.
C - Total Displacement of 2-byte Chain, same as A, except for 2-byte string.
D - Total Displacement of 1-byte Chain, same as A, except for 1-byte string.

EXAMPLE OF AN ASSEMBLER EDITED TEXT RECORD AND ATTACHED WORKBUCKETS





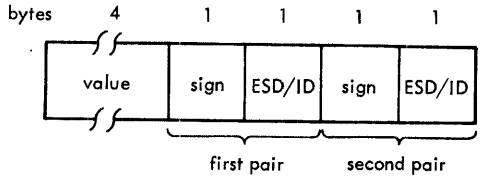
EVALUATION WORKBUCKET



Type ID - The first three bits have the binary value of 011, which indicates that this is an Evaluation Workbucket.
Value Field - The value field has two formats. One format is for evaluating

DC Fixed - or Floating-point constants. The other format is for evaluating an expression.

For Fixed- or Floating-point constants, the 8-byte value field contains the constant value evaluated in phase 21A. For expressions, the 8-byte value field will have the following format.



Value - The value resulting from the evaluation of an expression appearing in the operation field of a statement.

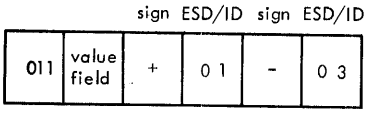
Sign - ESD/ID Field - The contents of this field depend upon whether the expression was absolute, simple relocatable, or complex relocatable.

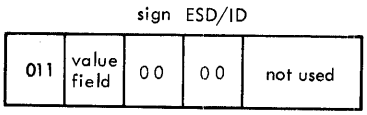
For absolute and simple relocatable expressions involving no unpaired terms, the first Sign-ESD/ID pair of bytes will be X'0000', and the second pair of bytes will not be used.

For simple relocatable expressions involving an unpaired term, the sign-byte of the first Sign-ESD/ID pair will indicate if the relocation factor should be added or subtracted from the contents of the value field. The ESD/ID byte of the first Sign-ESD/ID pair will contain the External Symbol Dictionary Identification number of the unpaired term. The second Sign-ESD/ID pair will contain X'0000'.

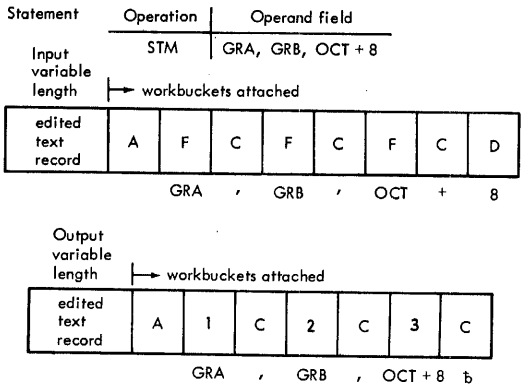
For complex relocatable expressions, there can be up to sixteen Sign-ESD/ID pairs. In any case, the last pair must contain X'0000' to terminate the scan. The Evaluation Workbucket is fixed length, 9 bytes. If there is more than one Sign-ESD/ID pairs, it will be necessary to create more evaluation workbuckets.

Example, showing two Sign-ESD/ID pairs.





Example, showing machine instruction use.



A - Statement workbucket
F - Symbol workbucket
1 - Evaluation workbucket 1
2 - Evaluation workbucket 2
3 - Evaluation workbucket 3

Value (Hexadecimal)	Description
FF	End of Statement Record Flag (E)
FE	End of Block
FD	No Evaluation Necessary Flag 'PUT'
FC	Keyword Flag (K)
FB	Positional Flag (P)
FA	Symbol Flag
F9	Continue Sublist Flag
F0	Sublist Flag
F8	End of Statement Field Flag (Y)
EB	End of Buffer Flag
EBFF	End of Data Set

APPENDIX G. MACRO GENERATOR VALUE ASSIGNMENT FOR EXPRESSION EVALUATION

	CHAR	DEC	HEX
Period	.	0	0
Right Parenthesis	)	1	1
Left Parenthesis	(	2	2
Subscripted Left Parenthesis	¢	3	3
Plus	+	4	4
Minus	-	5	5
Multiply (AST)	*	6	6
Divide (Slash)	/	7	7
Equal	EQ	8	8
Not Equal	NE	9	9
Less Than	LT	10	OA
Greater Than	GT	11	OB
Less Than or Equal to	LE	12	OC
Greater Than or Equal to	GE	13	OD
Not	NOT	14	OE
Or	OR	15	OF
And	AND	16	10
Hexadecimal Self-Defining Term	X"	34	22
Binary Self-Defining Term	B"	35	23
Decimal Self-Defining Term	DECINT	36	24
Character Self-Defining Term	CSD	37	25

	CHAR	DEC	HEX
Null Symbol & Evaluation Flag	NULLSYM	38	26
Character String	C"	39	27
SETA	SETA	40	28
SETB	SETB	41	29
SETC	SETC	42	2A
Comma	Comma	43	2B
Begin Substring	BEGSUB	44	2C
Begin Substring Operands	SUBOPE	45	2D
First Operand Completed	SUBCOM	46	2E
Second Operand Completed	SUBCLS	47	2F
Actual Internal Value Right Parenthesis Used Only on Sublist	ACT.)	48	30
Arithmetic Expression (absence indicates character expression)	AE	49	31
Blank	BLANK	50	32
Type Attribute Reference	T'	51	33
Length Attribute Reference	L'	52	34
Integer Attribute Reference	I'	53	35
Scale Attribute Reference	S'	54	36
Number Attribute Reference	N'	55	37
Count Attribute Reference	K'	56	38
Symbolic Parameter Reference	PAR	57	39
System List	SYSLIST	58	3A

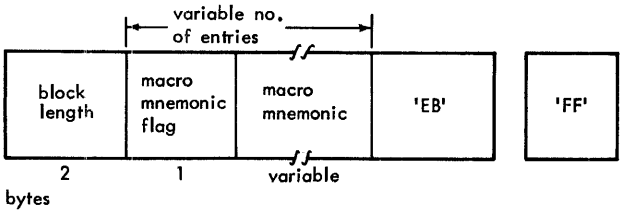
Type	Decimal Value	
P	0	
Z	1	
E	2	FLOATING
D	3	
K	4	
F	5	FIXED
G	6	
H	7	
S	8	
A	9	
V	10	
Y	11	
R	12	
W	13	
I	14	
C	15	
Q	16	
B	17	
J	18	
X	19	
M	20	These six types must be last on the list
T	21	
U	22	
O	23	
N	24	
U'	25	

APPENDIX I. TABLE FORMATS

Table Identification and Blocking Sizes

TABLE	ID	MINIMUM SIZE IN BYTES
Macro Mnemonic Name Table Segments (Unsubsetted)	LOCATION	100
Macro Mnemonic Name Table Segments (Undefined-Subsetted)	LOCATION	100
Macro Mnemonic Name Table Segments (Defined-Subsetted)	LOCATION	1000
Relevant Ordinary Symbol Table Segments	LOCATION	200
Macro Dictionaries (Unsubsetted)	LOCATION	256
Macro Dictionaries (Subsetted)	LOCATION	200
Literal Table Segments	X'09'	800
Symbol List Table Segments	X'01'	160
Symbol List Table Sections	X'02'	1000
External Symbol Dictionary Segments	X'03'	260
Literal Base Table	X'04'	131
Symbol Table Segments (Adjusted and Unadjusted)	X'05' and LOCATION	3000
Literal Adjustment Table	X'06'	84
Cross-Reference Table Records	X'07'	161
Relocation Dictionary Table Records	X'08'	181

MACRO (MNEMONIC) NAME TABLE SEGMENT



Block Length (2 bytes) - Hexadecimal length of block.

Macro Mnemonic Flag (1 byte)

Bit		Description
0-1	10	Undefined macro mnemonic
	11	Defined macro mnemonic

NOTE: All macro names in subsetted table produced by E2A are marked as "DEFINED".

2-4 Not used

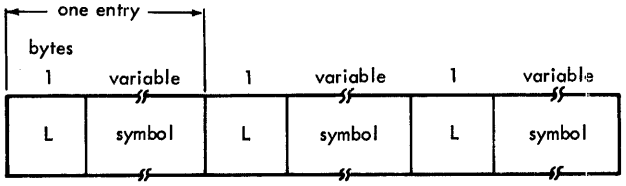
5-7 Actual length minus one of macro mnemonic

Macro Mnemonic - Macro mnemonic in character format (variable)

'EB' - End of Block Flag (1 byte)

'FF' - End of Data Flag (1 byte), the last block of macro names are indicated by a 1-byte End of Data Flag ('FF') following the End of Block Flag ('EB').

RELEVANT ORDINARY SYMBOL TABLE SEGMENT

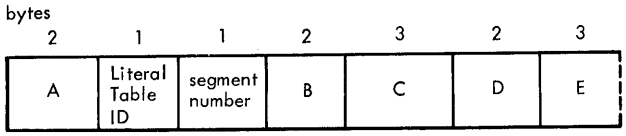


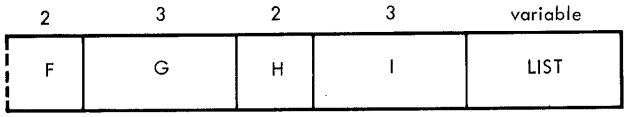
The Relevant Ordinary Symbol Table Segment is made up of a variable number of length and symbol entries.

L - The length of the following symbol in bytes.

Symbol - The character representation of the symbol.

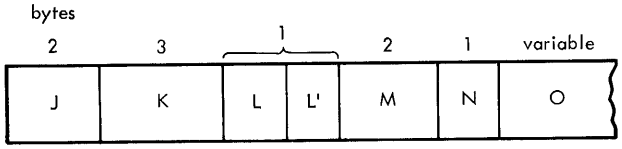
LITERAL TABLE SEGMENTS





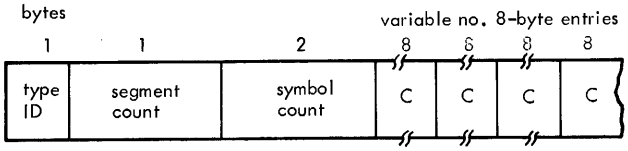
A - Next available location in literal table stack, initial value X'16'.
Literal Table ID - Literal table identification (09).
Segment Number - Not currently used by I/O package.
B - Pointer to first entry in 8-byte chain.
C - Displacement value of 8-byte chain (i.e., the total object length occupied by the literals in the 8-byte chain, including duplication factor and multiple constants).
D - Pointer to first entry in 4-byte chain.
E - Same as C, except as applicable to 4-byte chain.
F - Pointer to first entry in 2-byte chain.
G - Same as C, except as applicable to 2-byte chain.
H - Pointer to first entry in 1-byte chain.
I - Same as C, except as applicable to 1-byte chain.

List as follows:



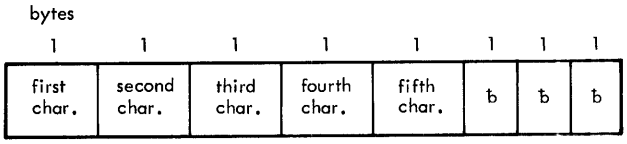
J - Same chain forward pointer.
K - Object displacement within chain.
L - Not used.
L' - *in literal switch.
M - Statement number. Present only if switch L' = 1. The statement number is obtained from a counter internal to the phase. Phase 9 generates the statement number in the same way.
N - Literal source length.
O - Literal source to the right of the equal sign.

SYMBOL LIST TABLE SEGMENTS



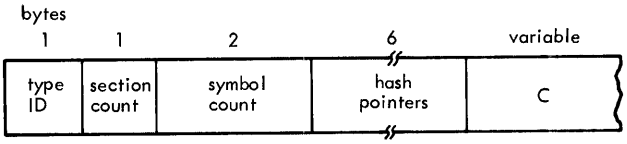
Type ID - Type Identification (01) indicates Symbol List Table Segment.
Segment Count - Not currently used by I/O package.
Symbol Count - Number of 8-byte symbol entries.
C - 8-byte symbol entries, including symbols used in symbolic length attribute references for the following:
1. CNOP operand
2. ORG operand
3. EQU operand
4. DC/DS/CCW length, scale, or exponent modifier expressions
5. DC/DS/CCW duplication factor expressions
6. Named CSECT, START, DSECT
7. Symbols in operand field of EXTRN statements

Symbol List Segment Entry (Sample 5-Character Symbol)



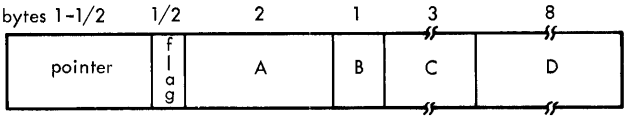
NOTE: Symbols are left-justified, followed by blanks (b) if there are less than eight characters

SYMBOL LIST TABLE SECTIONS



Type ID - Type Identification (02) indicates Symbol List Table Sections.
Section Count - Not used.
Symbol Count - Number of 16-byte symbol entries.
Hash Pointers (2 bytes per 48 symbol entry bytes) - Pointer to symbol address in symbol list table (see hash table).
C - 16-byte symbol entries.

SYMBOL LIST TABLE ENTRY



Pointer - Relative to the table origin, pointing to the next entry in the chain. The pointer is positioned for use after flag and adjustment code are marked out. It is abbreviated because

entries are aligned on double-word boundaries.

Flag - Zero when not yet evaluated, when equal to one, indicates value and length attribute are present (see Statement Workbucket).

A - Length attribute of symbol.

B - External Symbol Dictionary Identification.

C - Value of symbol.

D - 8-byte symbol entered in phase E7I (see Symbol List Segment Entry).

EXTERNAL SYMBOL DICTIONARY SEGMENTS

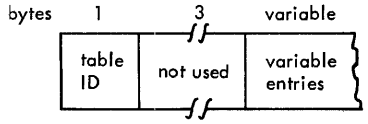
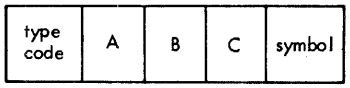


Table ID - X'03'

Variable Table Entries



Type Code:

HEX	MNEMONIC	DESCRIPTION
'00'	SD	Named CSECT or START instruction
'01'	LD	ENTRY symbol
'02'	ER	EXTRN symbol
'03'		Not used
'04'	PC	Private code unnamed control section
'05'	CM	Common (COM) control section
'06'		Pseudo registers not currently implemented
'07'	DS	Named DSECT
'08'	VC	V-type address constant symbol
'09'	BD	Unnamed (illegal) DSECT

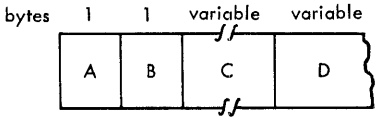
A - For all but type LD, ER, PR, and VC; current value of Location Counter for this control section. For LD, low-order byte contains ESD/ID for the control section in which the symbol is defined.

B - CLF for all but LD, ER, and VC; the ESD/ID of the control section. ESD/ID numbers are assigned sequentially downward, starting with 255 for named or unnamed DSECT entries (e.g., DS or BD). The numbers are assigned upwards beginning with 1 for SD, PC, and CM. For LD, this byte contains a flag that is set in phase 9I during processing of ENTRY (LD) symbols.

C - HLF for all but LD, ER, and VC; the highest value yet held by the Location Counter for this control section. This indicates the length in bytes of this control section. Filled in phase 8. For LD, this field contains the (adjusted) assembled address of the entry point symbol in the symbol field. Filled in phase 9I.

Symbol - The symbol in byte format, that is associated with this table entry.

SYMBOL TABLE SEGMENTS

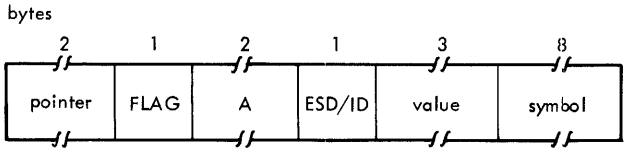


A - Table Identification ('05') indicates Symbol Table Segment.

B - Segment flag; AA if this is not the last segment or FF if this is the last or only segment.

C - Hash table 2-byte address entries.

D - Item Entry



Ptr - Forward pointer to next entry in same chain, if applicable.

Flag - Identical to the second byte of the flag used in the Statement workbucket. However, it is created in phase 9, and not copied from the Statement workbucket.

A - The length attribute of the symbol located in the symbol field.

ESD/ID - External Symbol Dictionary Identification.

Value - In phase 9, this is the relative address. In phase 9I, this is adjusted to the final assembled address.

Symbol - Same as shown for Symbol List Table Segments.

LITERAL BASE TABLE

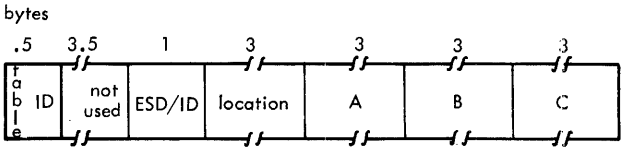
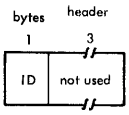


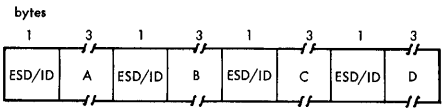
Table ID - X'4', indicates Literal Base Table.

ESD/ID - The ESD/ID number of the control section where the literal pool is located.

Location - This is the relative address obtained from the Statement workbucket attached to the associated LTORG statement.
A - The total object length of the literals comprising the 8-byte string in the associated literal pool.
B - Same as A, except as applicable to the 4-byte string.
C - Same as A, except as applicable to the 2-byte string.

LITERAL ADJUSTMENT TABLE

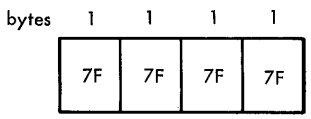




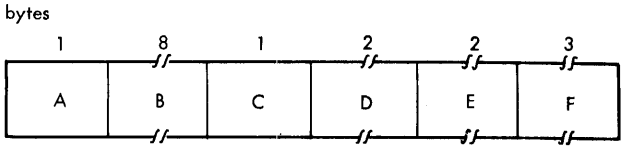
ID - Table identifier ('60') indicates Literal Adjustment Table.
ESD/ID - External Symbol Dictionary Identification of the 3-bytes that immediately follow this byte. Typical of four shown.
A - The adjusted assembler address of the beginning of the 8-byte string of literals whose pool is described by this table.
ESD/ID - Same as previously described for this table.
B - Same as A, except as applicable to the 4-byte string.
ESD/ID - Same as previously described for this table.
C - Same as A, except as applicable to the 2-byte string.
ESD/ID - Same as previously described for this table.
D - Same as A, except as applicable to the 1-byte string.

NOTE: There is one such table for each LTORG or END assembler instruction in the program.

Trailer: Indicates the end of the Literal Adjustment Table.

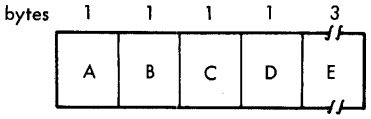


CROSS REFERENCE TABLE RECORDS



A - Table Identification ('07') indicates Cross Reference Table Record.
B - Symbol as described in Symbol List Segment Entry.
C - Flag; 0 Base symbol (e.g., initial definition)
1 Reference to symbol
D - Statement number associated with the statement where the symbol was defined.
E - The length attribute of the symbol.
F - The value attribute of the symbol (usually an address).

RELOCATION DICTIONARY TABLE RECORDS



A - Table Identifier ('08') indicates Relocation Dictionary Table Record.
B - Position ESD/ID number of the control section where the address constant is located.
C - Relocation ESD/ID number of the control section where the symbol is defined.
D - Flag

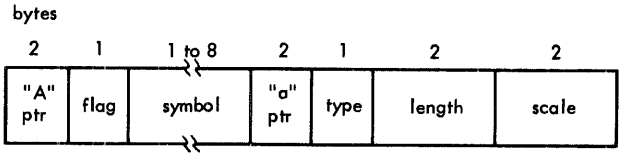
Bit		Description
1-3		Not used.
4	0	A- or Y-type constant and second operand of CCW.
	1	V-type constant.
5-6		Length constant in bytes:
	00	1 byte.
	01	2 bytes.
	10	3 bytes.
	11	4 bytes.
7	0	Relocation factor added .
	1	Relocation factor subtracted.
8	0	Next load constant has different position or relocation ESD/ID.
	1	Next load constant has same position or relocation ESD/ID.

NOTE: This bit is set during the post-process phase.

E - Assembler assigned address of a symbol used in an A-, Y-, or V-type constant, or second operand of CCW.

PHASE 4 LOCAL DICTIONARIES

OPEN CODE ORDINARY SYMBOLS



This format is for DC/DS Relevant Ordinary Symbol Table entries, or symbol whose attributes are referenced in conditional assembly statements.

"A" ptr - Same as macro mnemonic name entry.

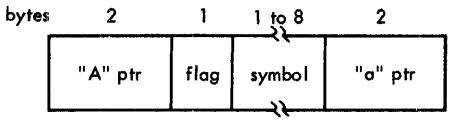
Flag

Bit		Description
0	0	Synonym (part of a chain).
	1	End of the chain.
1-4	1000	Tags.
	1001	Parameters.
	1010	Symbols.
	1100	Local A variables.
	1101	Local B variables.
	1110	Local C variables.
5-7		Length (L-1) of BCD entry.

Symbol - The symbol in character format.
"a" ptr - Same macro mnemonic name entry.
Type - Type attribute (see Table of Parameter Type Attributes).
Length - Length attribute.
Scale - Scale attribute:

Bit		Description
0	0	Positive.
	1	Negative.
1-15		Scale attribute.

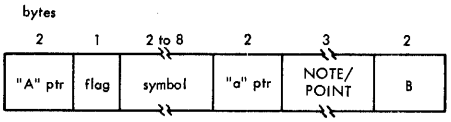
RELEVANT ORDINARY SYMBOLS



This format is for entries from Relevant Ordinary Symbol Table.

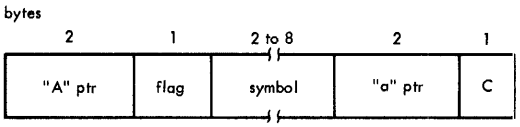
"A" ptr - Same as macro mnemonic name entry.
Flag - Same as open code ordinary symbols.
Symbol - Same as open code ordinary symbols.
"a" ptr - Same as macro mnemonic name entry.

SEQUENCE SYMBOLS



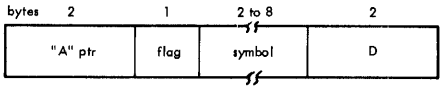
"A" ptr - Same as macro mnemonic name entry.
Flag - Same as open code ordinary symbols.
Symbol - Same as open code ordinary symbols.
"a" ptr - Same macro mnemonic name entry.
Note/Point - The pointer to the block in which the fully edited text for the statement named by the sequence symbol begins.
B - The position of the beginning of the sequence symbol fully edited text relative to the beginning of the block in which it is located.

LOCAL SET VARIABLE SYMBOLS



"A" ptr - Same as macro mnemonic name entry.
Flag - Same as open code ordinary symbols.
Symbol - Same as open code ordinary symbols.
"a" ptr - Same as macro mnemonic name entry.
C - The declared dimension of the local set variable symbol. Will be zero if the symbol is undimensioned.

MACRO PROTOTYPE SYMBOLIC PARAMETERS



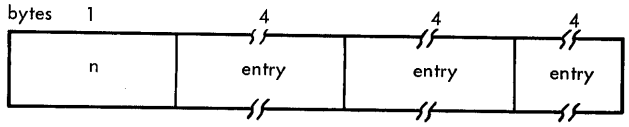
This format is for keyword and positional type.

"A" ptr - Same as macro mnemonic name entry.
Flag - Same as open code ordinary symbols.
Symbol - Same as open code ordinary symbols.
D - The operand position number assigned to the symbolic parameter.

PHASE 5 DICTIONARY ENTRY

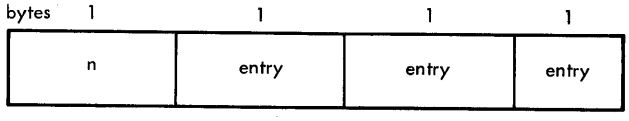
SET VARIABLES

SETA Variable



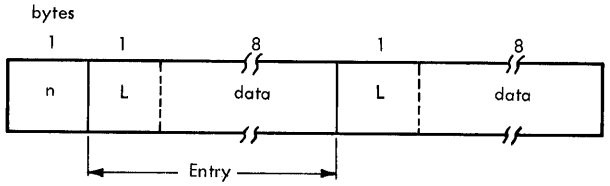
Length of complete entry is equal to 4n + 1. There are four bytes per sub-entry. n = SET variable dimension. This byte is not present if n = 1, i.e. an undimensioned SET variable.

SETB Variable



Length of complete entry is equal to n/8 + 1 (i.e., each SETB variable is evaluated in one bit. 0 equals false, and 1 equals true). There is one byte per entry; each entry contains up to 8 SETB evaluations. n = same as SETA variable.

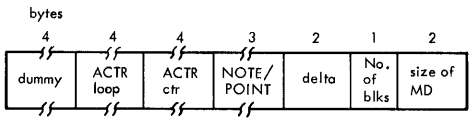
SETC Variable



Length of entry is equal to 9n + 1. There are nine bytes per entry.

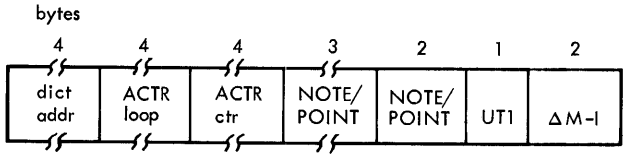
n Same as SETA variable.
L Length of character string (data), true length.

MACRO DICTIONARY HEADER



Header attached to subsetted dictionaries output by phase E4S.

Dummy - Not used.
ACTR Loop - The A counter loop limit (i.e., the number of passes through the loop).
ACTR ctr - The A counter current loop count (i.e., the number of times the loop has been passed).
Note/Point - Location of block on SYSUT3 in which the macro definition fully-edited text begins.
Delta - Location of the prototype fully edited text relative to the beginning of the block.
No. of blks - The number of blocks of 256 bytes on SYSUT3 in which the dictionary is contained.
Size of MD - The size of the macro dictionary in bytes.

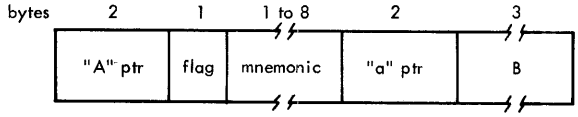


Header as modified during Phase 5 processing of macro instruction.

Dict Addr - The location of the next higher level local dictionary. If the macro being processed is not an inner macro, this pointer points to the open code local dictionary.
ACTR Loop - Same as macro dictionary header.
ACTR ctr - Same as macro dictionary header.
Note/Point (3 bytes) - The location of the block in which the end-of-macro instruction record is located. (2 bytes) - The location of the end of end-of-macro instruction record relative to the beginning of the block in which it is located.
UT1 - A switch to indicate:
0 Input is from SYSUT3.
1 Input is from SYSUT2.
ΔM-I - Delta, the location of the beginning of the macro instruction text, relative to the block in which it is located.

PHASE 4 GLOBAL DICTIONARY

MACRO MNEMONIC NAME ENTRY



"A" ptr - Big A pointer, backwards chain pointer to synonym

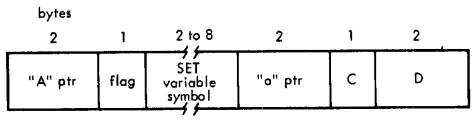
Flag

Bit		Description
0	0	Normal global variables.

Bit	Description
1	Obsolete global variables (global variables which have been declared apart from the current part of the source deck being processed, such as macro definition or mainline program).
	NOTE: Bit zero is used for global variables only
1-4	0000 Op codes. 0001 Pseudo op codes. 0010 Extended mnemonic. 0011 Macro names. 0100 Global A variables. 0101 Global B variables. 0110 Global C variables.
5-7	Length (L-1) of BCD entry.

Mnemonic - The macro name in byte format.
"a" ptr - Little "a" pointer to a location in the phase 5 dictionary that will contain the subsetted entry for this mnemonic.
B - Note/Point Address. Until the dictionary associated with this mnemonic is subsetted, this entry will contain the NOTE'd location of the beginning of the fully edited text for the macro definition associated with this mnemonic. After the local dictionary associated with this mnemonic has been subsetted, this field contains the NOTE'd location of the subset dictionary which will in turn contain the NOTE'd position of the macro definition fully-edited text for this mnemonic.

GLOBAL SET VARIABLE SYMBOL ENTRY

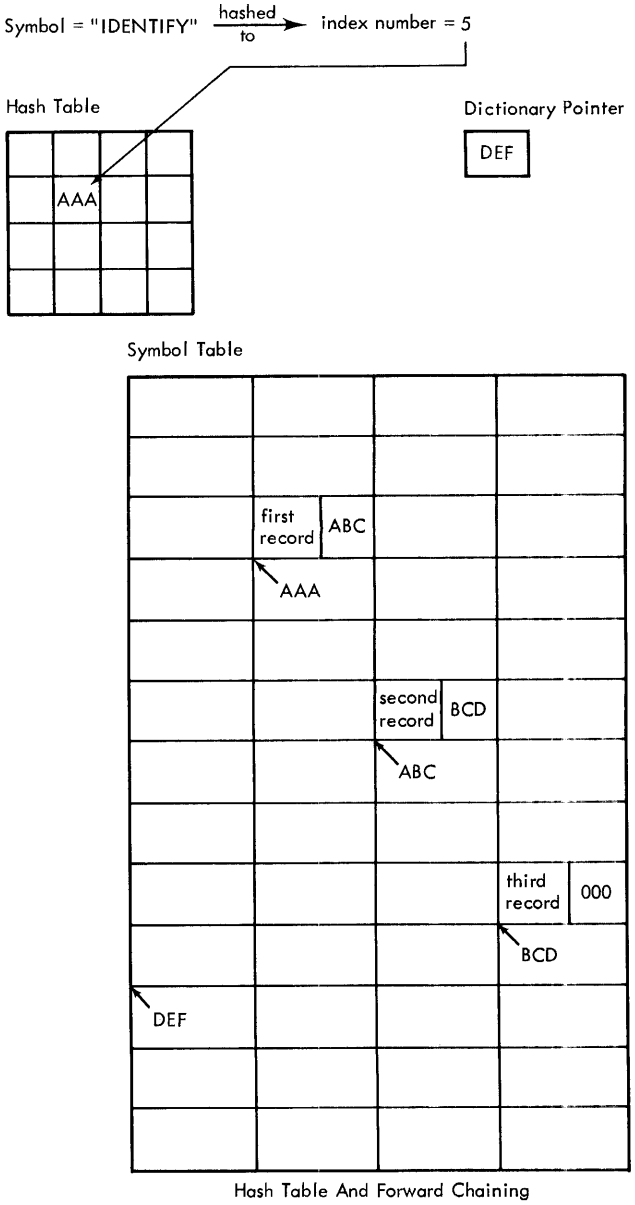


"A" ptr - Same as macro mnemonic name entry.
Flag - Same as macro mnemonic name entry.
Set Variable Symbol - The set variable symbol in byte format.
"a" ptr - Same as macro mnemonic name entry.
C - Dimension, the declared set variable dimension. Will be zero if undimensioned.
D - Activity bit pointer, points to a unique bit which when on indicates that the global symbol has already been defined within the macro or open code currently being processed (i.e., duplicate local definition).

APPENDIX J. HASH TABLE

A hash table is used by the assembler for inserting or locating variable or fixed length record entries in symbol tables and macro dictionaries. A hash table consists of space reserved for fixed length address entries (called pointers) which point to locations in the dictionaries/tables. The range of the hash table is the number of such pointers that can be placed in the reserved space. When it is desired to enter a symbol in the dictionary (e.g., enter a Global symbol declaration) or locate an entry in the dictionary (e.g., to obtain the relative address of a symbol) the associated symbol must first be randomized to produce an index number (a technique called hashing). The randomizing algorithm is such that the resulting index number will be a whole number between zero and the hash table range minus one. This index number is then used to index into the hash table and inspect the associated pointer (address entry) in the hash table. This entry will be zero until a symbol record entry, randomizing to this index number, has been entered in the dictionary/table. Records are entered in the dictionary sequentially and a dictionary pointer, containing the next available address in the dictionary, is used for inserting records in the dictionary. Several different symbols (called synonyms) may randomize to the same index number. Because this index number points to an associated entry in the hash table where only one address can be stored, chaining must be used to enter or locate the synonym records. Two types of chaining are used by the assembler, they are; 1) forward chaining, used for symbol and symbol list tables, and 2) backward chaining, used for the macro dictionary.

To enter records in the symbol and symbol list tables, the symbol whose record is to be entered is hashed to obtain an index number. This number is used to point to the associated address entry in the hash table. The hash table entry is inspected. It will be zero if no other symbols have yet hashed to the same index number (i.e., this is the first symbol record entry for this index number). If this is not the first entry of a record whose symbol hashed to this index number, the hash table will always contain the address of the first symbol record entered in this chain. At that address will be a symbol record containing a pointer

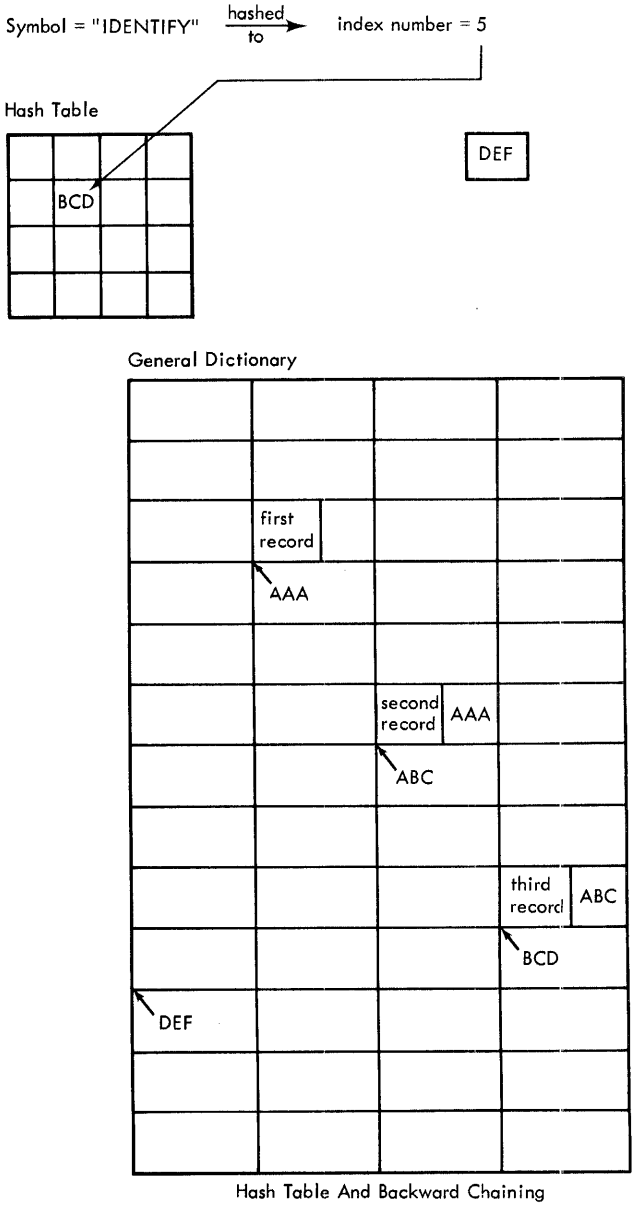


Hash Table And Forward Chaining

field. This pointer field will have either a chaining address pointing to the location of the next record in the chain or it will be zero, which indicates that this is the last (or only) record in the chain. If the pointer field contains a chaining address, the next record in the chain will have a pointer field the same as just described for the first. If this pointer field is zero, it indicates that this is the last record in the chain. If it is not zero, the next record in the chain will have to be inspected as was this record, and so on until the last record in the chain is encountered (zero pointer field), or a duplicate record is encountered. If a duplicate record is encountered, the scan is terminated and the record is not entered. If a duplicate record is not encountered before the zero pointer field, the dictionary pointer with the next available storage address is obtained. This address is placed in the pointer field of the previous record that contained a zero pointer field, the record is stored in the indicated address, and the dictionary pointer is updated to reflect the next available storage address, by the length of the record just entered.

The procedure used to locate records in the symbol and symbol list tables is the same as entering, except when the compared records are equal, the pertinent information is extracted, or the value information is inserted, as the case may be at the time. If no duplicate entry exists during this procedure, the symbol is either not defined previously, or undefined, depending on the table being processed.

To enter records in the Macro dictionary, the symbol whose record is to be entered is hashed to an index number. This index number is used to point to an associated address in the hash table. The hash table entry is inspected. It will be zero if no other symbols have hashed to this same index number. If this is not the first entry, the hash table will always contain the address of the last symbol record entered (i.e., the most current entry). If the hash table contains an address, the record at that address will have a chain pointer field containing the address of the previous entry in the chain, or no pointer field at all, if it is the first entry. The chain will be scanned, starting with the last entry made, and continuing through until either a duplicate record, or the first record is encountered. If a duplicate record in encountered before the first entry in the chain, the scan is terminated and the record is not entered. If a duplicate entry is not encountered before the first entry, the address of the last record in the chain is obtained from the hash table and placed in the pointer field



Hash Table And Backward Chaining

of the record being entered. The address of the dictionary pointer is obtained and replaces the address just extracted from the hash table. The record is stored at the address now indicated by the hash table. The dictionary pointer is updated to indicate the new available storage address, by the length of the record just stored.

The procedure to locate records in the macro dictionary is the same as entering, except when the compared records are equal, the pertinent information is extracted, or the value information is inserted, as the case may be.

PHASE	ASM	MAC	INP	Phase Logic	GETMAIN Minimum	System Functions
E1	470	540	240	2490	2620	4170
E2	470	540	240	5870	2620	4170
E2A	470	540	240	1520	2880	3310
E3	470	540	-	5810	2620	3310
E3A	470	540	-	4480	2620	3310
E4P	470	540	-	4860	3840	3310
E4M	470	540	-	4770	3840	3310
E4S	470	540	-	1210	4350	3310
E5P	470	540	-	4780	3530	3310
E5	470	540	-	4760	3330	3310
E5A	470	540	-	4620	3330	3310
E5E	470	540	-	600	3330	3310
		RTA	RTB			
07/07A	470	1400	-	2880	1780	3610
07/07B	470	1400	-	2820	1780	3610
07I	470	1400	-	1060	1160	3310
08/08A	470	1400	-	2680	2980	3610
08/08B	470	1400	-	2680	2920	3610
09	470	1400	350	4640	3930	3770
09I	470	1400	350	3380	4500	3770
10	470	1400	350	4160	4450	3770
10B	470	1400	350	7330	1200	3770
21A	470	1400	350	5850	1200	3770
21B/C	470	1400	350	6570	400	3800
21B/D	470	1400	350	6570	400	3800
PP	470	1400	350	5090	3000	3770
DI	470	1400	350	5910	-	3770

APPENDIX L. CONTROL PROGRAM SERVICES

Macro	Purpose
CALL	Passes control from a program, load module, or segment of an overlay module to a specified entry point in another program.
CHECK	Waits (if necessary) for the completion of a read or write operation and detects errors and exceptional conditions.
CLOSE	Closes a data-control-block after a data set has been processed.
DCBD	Generates a DSECT statement that provides a symbolic name for the fields within a data control block.
DELETE	Indicates to the supervisor that an in-storage copy of a load module (previously acquired by a LOAD macro-instruction) is no longer required. The storage occupied by that load module is freed for other uses.
FIND	Places the address of the first block of a specified partitioned data set member in the indicated data-control-block.
FREEMAIN	Releases one or more areas of main storage previously acquired through one or more GETMAIN macro-instructions.
GETMAIN	Requests the supervisor to allocate one or more areas of main storage for assembler use.
LINK	Passes control from one load module to a specified entry point in another load module, keeping both modules resident in main storage.
LOAD	Acquires a specified load module and causes the supervisor to retain the module for use by the task using the LOAD. If a copy of the load module is not currently available in main storage, one is fetched.

Macro	Program
NOTE	Requests the relative position within a volume of the block just read, or written, for the direct retrieval of the block at a later time.
OPEN	Initializes one or more data-control-blocks so that their associated data sets can be processed.
POINT	Alters the sequential processing of a data set such that the next read or write operation will take place at a previously NOTEd position.
READ	Retrieves the requested block from an input data set and places it in a main storage area.
RETURN	Indicates normal termination of a task and returns control to a higher level program or task, or to the control program.
SEGWT	Causes a specified segment to be loaded into main storage.
TIME	Provides the time of day in Register 0 and the current date in Register 1.
TCLOSE	Temporarily closes a data-control-block after a data set has been processed.
WRITE	Transfers a block from the user's main storage area to a physical sequential data set.
XCTL	Passes control from the load module in which it appears to the load module it specifies. Only the new module remains in main storage.

PHASE	SYSUT1	SYSUT2	SYSUT3	SYSIN	SYSLIB	SYSPUNCH	PHASE LOGIC	
E1	OPEN WRITE CHECK	OPEN	OPEN	OPEN READ CHECK	OPEN		GETMAIN LOAD (INP)	
E2/E2A	WRITE CHECK NOTE TCLOSE	READ WRITE CHECK POINT TCLOSE		READ CHECK CLOSE	FIND READ CHECK NOTE POINT CLOSE		DCBD DELETE (INP) GETMAIN FREEMAIN	E2 E2A
E3/E3A	READ CHECK	WRITE CHECK NOTE TCLOSE	WRITE CHECK NOTE TCLOSE				FREEMAIN	E3
E4P/E4M/ E4S	WRITE CHECK NOTE TCLOSE	READ CHECK TCLOSE	READ WRITE CHECK NOTE POINT TCLOSE				GETMAIN FREEMAIN GETMAIN FREEMAIN	E4P E4M E4S
E5P/E5/ E5A/E5E	READ CHECK NOTE POINT TCLOSE	WRITE CHECK TCLOSE	READ CHECK POINT TCLOSE				GETMAIN FREEMAIN FREEMAIN	E5P E5 E5E
07/07A/ 07B	READ WRITE CHECK NOTE POINT	READ CHECK TCLOSE	WRITE CHECK TCLOSE				GETMAIN FREEMAIN SEGWT CALL CALL	07 07A 07B
071	READ CHECK WRITE NOTE POINT				SYSPRINT		GETMAIN FREEMAIN SVC/0	
08/08A/ 08B	READ WRITE CHECK NOTE POINT	WRITE READ CHECK TCLOSE	READ WRITE CHECK TCLOSE				GETMAIN FREEMAIN SEGWT CALL CALL	08 08A 08B
09	WRITE CHECK NOTE	READ CHECK TCLOSE	READ CHECK TCLOSE		OPEN	OPEN WRITE CHECK	FREEMAIN	
091	READ POINT CHECK WRITE NOTE				WRITE CHECK OPEN	WRITE CHECK OPEN	GETMAIN FREEMAIN	
10	READ CHECK WRITE POINT	WRITE CHECK READ TCLOSE	READ CHECK WRITE TCLOSE				GETMAIN FREEMAIN	
10B	TCLOSE						GETMAIN FREEMAIN	
21A		WRITE CHECK READ TCLOSE	READ CHECK WRITE TCLOSE				GETMAIN FREEMAIN	
21B/21C/ 21D	WRITE CHECK NOTE	READ CHECK	READ CHECK		WRITE CHECK	WRITE CHECK	GETMAIN FREEMAIN TIME LINK	
PP	READ CHECK NOTE POINT	READ CHECK WRITE CLOSE	READ WRITE CHECK CLOSE		WRITE CHECK	WRITE CHECK		
DI	READ CHECK POINT CLOSE NOTE				WRITE CHECK CLOSE	CLOSE		

	Entry Type															
	SYSNDX	Parameter Number	NDX	L	4 bytes Character Value	4 bytes Binary Value										
	SYSECT	Not Used	CSECT	L	BCD Name											
1	Name field of Character string Operand	T	CHAR	K' L	Name or Character String											
2	Self-defining Term (Hex, Binary, Decimal)	T	HBD	3 bytes Binary			'c'	K' L	Character Representation							
3	Self-defining character	T	CSD	3 bytes Binary			'c'	K' L	Character Representation							
4	Symbol	T	SYM	5 bytes Attributes of symbol			'c'	K' L	Symbol Name							
	Sublist	T	SUB	2 bytes Tot. L	2 bytes K'	N'	(	2 bytes L_1	Parameter Entry Type 1, 2, 3, or 4 above	,	2 bytes L_2	Parameter Entry Type 1, 2, 3 or 4 above	)	N'		

T = Type attribute

L = Length of following field (For HBD, CSD, CHAR and SYM L = K')

'c' = character flag — a character string follows

Tot. L = Total length of sublist entry

K' (not sublist) = Number of characters in operand (excluding commas)

K' (sublist) = Number of characters between outer commas in a sublist

N' = Number of operands in a sublist

N' = 1 if the operand is a sublist

N' = 0 if the operand is omitted

The terms in this glossary are defined relative to their use in this publication only. These definitions may differ from those in other publications.

Assemble: To prepare an object language from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

Attributes: There are six kinds of attributes in the macro generator: Type, Length, Scaling, Integer, Count, and Number. The assembler portion processes only the length attribute.

Basic Partitioned Access Method (BPAM): A method of storing and retrieving sequences of data "members" belonging to a data set stored on a direct-access device; the data set contains a directory that relates the member name with the address where the sequence starts.

Basic Sequential Access Method (BSAM): A method of storing and retrieving physical blocks of data from a sequentially organized data set.

Concatenation: The process of linking together, chaining, or joining.

Control Program: A collective term referring to all control routines of the operating system.

Control Section: The smallest separately relocatable unit of a program; that portion of text specified by the programmer to be an entity, all elements of which are to be loaded into contiguous main storage locations.
A block of coding that can be relocated, independently of other coding, at load time without altering or impairing the operating logic of the program; it is identified by a CSECT or START instruction.

Data Control Block (DCB): A region in storage used for communication between the source program, the control program, and the access routines.
A control block through which the information required by access routines to store and retrieve data is communicated to them.

Data Set: A named collection of data.

Direct Access: Retrieval or storage of data by a reference to its location on a volume, rather than relative to the previously retrieved or stored data.

External Symbol Dictionary (ESD): Part of an object or load module produced by the assembler that identifies the external symbols and entry-point symbols referenced by and defined within the module.

Function Block: A portion of the coding that constitutes a logical element.

Global Dictionary: A table containing macro mnemonics and global variable symbols.

Global Variable Symbols: Symbols that communicate values between statements in one or more macro-definitions and statements outside macro definitions.
Global SET symbols are the only global variable symbols.

Hashing: Processing a symbol in order to arrive at an address that falls between two limits.

Hash Table (HT): A table of addresses that point to entries in a dictionary.

Inner Macro-instruction: A macro-instruction used as a model statement in a macro-definition.

Linkage Editor: A program that produces a load module by transforming object modules into a suitable format.

Literal: A representation of a constant which is entered into a program by specifying the constant in the operand of the instruction in which it is used. The assembler stores the value specified by the literal in a literal pool, and places the address of the storage field containing the value in the operand field of the assembled source statement.

Literal Pool: A portion of the object program containing literals processed by the assembler.

Load Module: A module in a format suitable for loading into main storage by the control program.

Local Dictionary: A table containing local variable symbols and sequence symbols.

Local Variable Symbols: Symbols that communicate values between statements in the same macro-definition, or between statements outside macro-definitions. The following are local variable symbols:
1. Symbolic parameters
2. Local SET symbols
3. System variable symbols

Logical Element: An intermediate level of logic, comprising a group of function blocks, which fulfills one of the specific objectives of a major component.

Logical Record: A record from the standpoint of its content, function, and use rather than its physical attributes; i.e., one that is defined in terms of the information it contains. (Contrasted with Physical Record.)

Macro-definition: A set of statements that provides the assembler with the mnemonic operation code and the format of the macro-instruction, and the sequence of statements the assembler generates when the macro-instruction appears in the source program.

Macro-instruction: A source-program statement for which the assembler generates a sequence of assembler language statements for each occurrence of the macro-instruction. Three types of macro-instructions may be written:
1. Positional - operands in fixed order
2. Keyword - operands in variable order
3. Mixed-mode - combination of above

Macro-instruction Prototype: The second statement of every macro-definition; it specifies the mnemonic operation code and the format of all macro-instructions that refer to the macro-definition.

Main Storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

Major Component: The largest describable logical division of the assembler program, i.e., a phase.

Model Statements: The macro-definition statements from which the desired sequences of assembler language statements are generated.

Module: One or more relocatable units of a program processed in one execution. The input to, or output from, a single execution of the assembler; a program unit that is discrete and identifiable with respect to the assembling process.

Object Program: A machine-language program which is the output after translation from the source program.

Operating System/360: A modular and device independent operating system requiring direct-access-storage-device residence; minimum core storage requirement is 32K.

Ordinary Symbol: One alphabetic character followed by zero through seven alphameric characters.

Outer Macro-instruction: A macro-instruction that is not used as a model statement in a macro-definition.

Overlay: A section of a program loaded into main storage, replacing all or part of a previously loaded section.

Physical Record. A record from the standpoint of the manner or form in which it is stored, retrieved, and moved; i.e., one that is defined in terms of physical qualities, or is meaningful with respect to access. (Contrasted with Logical Record.)

Pointer: An address used to point to a table or file entry.

Prototype Statement: See Macro-instruction Prototype.

Pseudo-operation Code: A hexadecimal one-byte code assigned to all assembler instructions (pseudo-ops) by programming systems for internal use.

Record: A general term for any unit of data that is distinct from all others when considered in a particular context.

Relevant Ordinary Symbol: An ordinary symbol relevant to the macro generator, i.e., for macro-instruction generation or conditional assembly.

Relocation Dictionary (RLD): Part of an object or load module that identifies all relocatable address constants in the module.

Relocation Identifier: A two-digit hexadecimal value used to associate an

instruction with a control section for relocation purposes.

SET Symbols: Variable symbols defined in the name field of SETx statements.

Significant Comma: A comma that delimits (defines end of field) the parameters of the operand list or the elements of a sub-list.

Significant Format: The source statement image with variable symbols, etc., replaced by flags and pointers.

Source Program: A series of statements in the symbolic language of the assembler that is input to the translation process.

Synonyms: Two or more symbols that result in the same address when they are hashed by a hashing routine.

System Macro-instructions: Macro-instructions that correspond to macro-definitions prepared by IBM.

Task: A unit of work for the central processing unit from the standpoint of the control program; the basic multi-programming unit under the control program.

Test Translator (TESTRAN): A facility that allows various debugging procedures to be specified in assembler language programs.

Utility Data Set: In Operating System/360, a data set reserved for intermediate results.

Variable Symbol: A type of symbol that is assigned different values by either the programmer or the assembler, thus allowing different values to be assigned to one symbol. There are three types of variable symbols: symbolic parameters, system variable symbols, and SET symbols. Variable symbols consist of an ampersand followed by an ordinary 1 - 7 character symbol.

Work Bucket: Data added to a text record to hold intermediate and/or final results of processing that record.

READER'S COMMENT FORM

IBM S/360 Operating System
Assembler (32K) Program Logic Manual

Form Y26-3598-0

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? _____
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other _____

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Y26-3598-0

fold

fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. 452

fold

fold

IBM /360 Printed in U.S.A. Y26-3598-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601

Y26-3598-0

IBM S[redacted]60 Printed in U.S.A. Y26-3598-0

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York