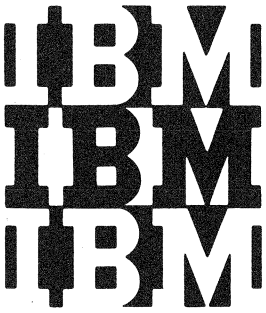


File No. S360-21
Order No. GY26-3700-2

Program Logic

IBM System/360 Operating System Assembler (F)

Program No. 360S-AS-037

This publication describes the internal logic of the F Assembler for the IBM System/360 Operating System. It is intended for use by persons involved in program maintenance and by system programmers who are altering the program design.

Preface

This manual describes the internal logic of the IBM System/360 Operating System F Assembler. The introduction gives the purpose of the assembler and summarizes the system required to operate this assembler. The organization of the F Assembler follows. The bulk of the manual is devoted to detailed descriptions of the operating phases of the F Assembler.

Effective use of this document is based on an understanding of the latest versions of the following manuals:

IBM System/360 Operating System, Assembler Language, Order No.GC28-6514

IBM System/360 Operating System, Principles of Operation, Order No.GA22-6821

IBM System/360 Operating System, Linkage Editor and Loader, Order No.GC28-6538

IBM System/360 Operating System, Supervisor and Data Management Services, Order No. GC28-6646

IBM System/360 Operating System, Concepts and Facilities, Order No. GC28-6535

IBM System/360 Operating System, Supervisor and Data Management Macro Instructions, Order No.GC28-6647

IBM System/360 Operating System, Job Control Language Reference, Order No. GC28-6704

IBM System/360 Operating System, Assembler F Programmer's Guide, Order No.GC26-3756

Third Edition (December, 1970)

This is a major revision of, and obsoletes, GY26-3700-1 and Technical Newsletter GY33-8028. This edition reflects the following changes:

- The SYSTERM data set added
- Changed logic for OPSYN handling
- The WXTRN instruction added

Other changes to the text and small changes to illustrations are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to release 20.1 of IBM System/360 Operating System and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to specifications herein; before using this publication in connection with the operation of IBM Systems, consult the latest SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Nordic Laboratory, Publications Development, Box 962, s-181 09 Lidingö 9, Sweden.

Figures

Tables

Flowcharts

PURPOSE OF THE ASSEMBLER

The assembler is designed to translate a source program coded in IBM System/360 Operating System Assembler Language into a relocatable machine language object program. The assembler also assigns relative storage locations to instructions and other program elements and performs auxiliary assembler functions designated by the programmer. The assembler performs its functions in two principal sections: the macro generation and conditional assembly section (Phases F1, F2, and F3) and the assembly section (Phases F7, FI, F8, and FPP).

The object programs produced by the assembler are in the format required by the linkage editor. The linkage editor produces load modules, following source program assembly, which are executable under the control of the operating system.

Several output options are available for the assembler. The programmer specifies the device for the object modules, etc. He may request an assembly listing, a cross-reference table of symbols as part of the listing and the insertion of a special source symbol table in the object module to facilitate TESTRAN. See Appendix A for details.

SYSTEM ENVIRONMENT

The IBM System/360 Operating System consists of a control program and several processing programs. The control program governs the order in which the processing programs are executed, allocates resources, 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 usually provided by IBM to assist the system user, plus problem programs written by the user and incorporated as part of the system. The assembler is a language translator, one of the processing programs of the IBM System/360 Operating System. See Figure 1.

The assembler can operate on IBM System/360 Models 30, 40, 50, 65, 75, or 85 with at least 64K storage. It requires only the standard instruction set.

SYSTEM AND I/O REQUIREMENTS

These requirements are divided into three categories: internal (or main) storage, external storage devices (data sets), and control program services.

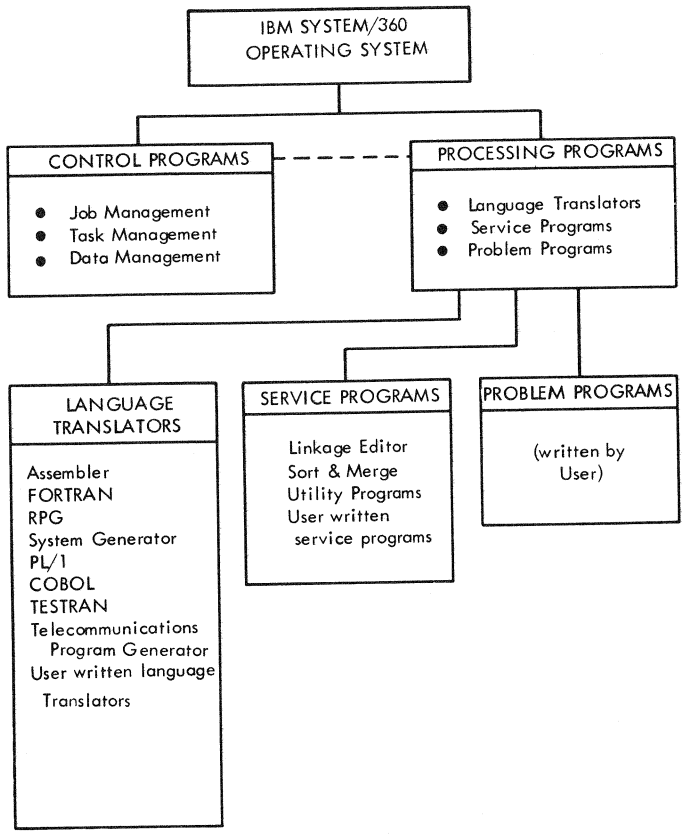


Figure 1. IBM System/360 Operating System

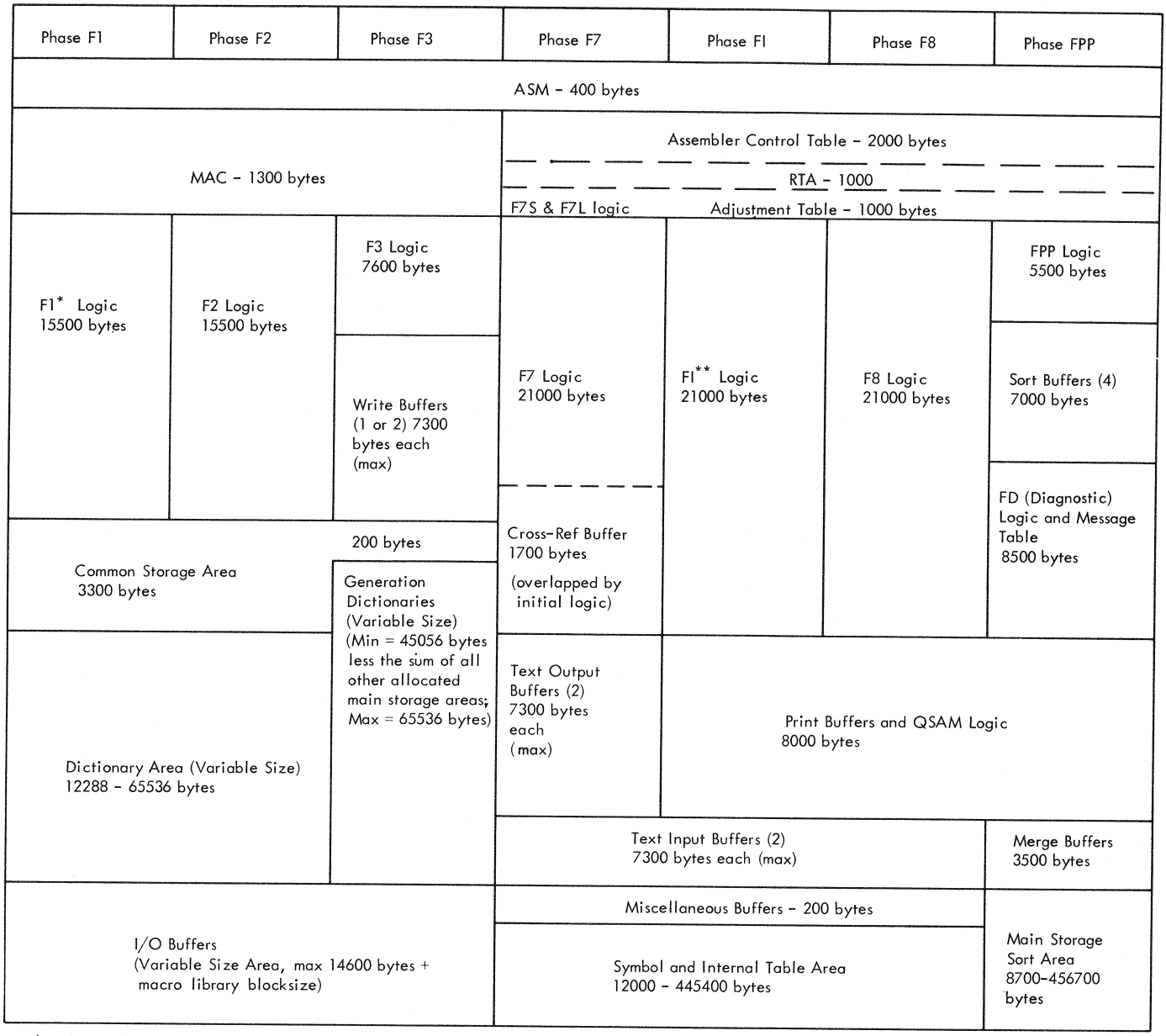
Main Storage

Main storage requirements include the minimum requirement of the Operating System Control Program. In addition, at least 45,056 bytes of contiguous core storage must be available for the assembler. Additional core storage will be used if available (see Table 1). The assembler is not reentrant.

Data Sets

The required data sets include one (SYSRES) containing the operating system components, which usually includes the assembler, three utility data sets (SYSUT1, SYSUT2, and SYSUT3), one input data set (SYSIN), and a data set (SYSLIB) containing system macro definitions and source coding to be called for through COPY assembler instructions. There are four optional output data sets as follows:

●Table 1. Storage Allocation



* NOTE: The F1 logic area is apportioned to the size of F2 logic for core storage management purposes. The F1 logic actually occupies only a fraction of this area.

** NOTE: The actual size of FI logic is approximately 3800 bytes. Part of the remaining area is used for storing tables. The text input buffers are not used by this phase. However, the large logic area and the text input buffer areas are maintained throughout the phase to avoid possible fragmentation of core by the Operating System and consequent unavailability for Phase F8.

1. Print data set for the ordinary listing (SYSPRINT).
2. Assemble-and-go data set (SYSGO).
3. Punch data set (SYSPUNCH).
4. Remote data set for diagnostic information (SYSTERM).

See Figure 2 and Table 2.

SYSRES. This is a Direct Access Storage Device (DASD) resident data set which contains the control program and other operating system components. The assembler program, itself a component of the operating system, is usually kept on this data set.

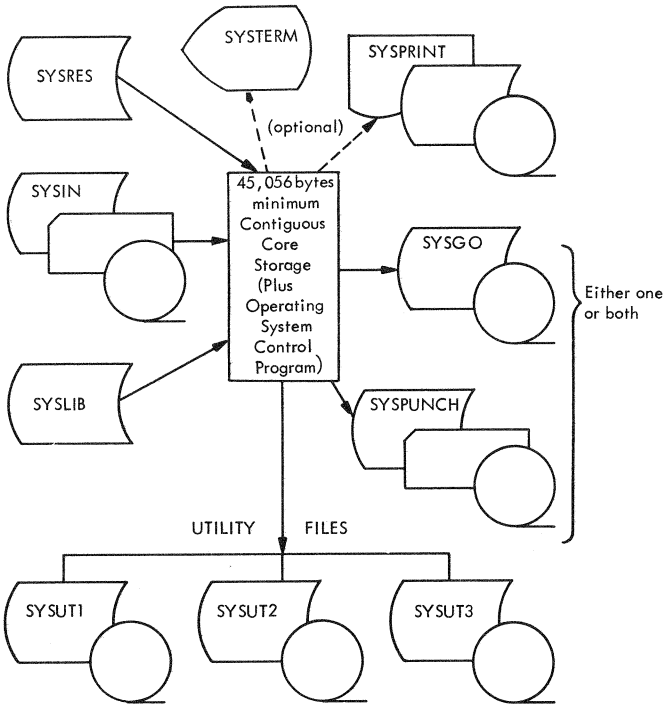
SYSUT1, SYSUT2, SYSUT3. These three utility data sets are used as intermediate external storage. Three DASD logical files, three magnetic tape units, or any combination of the DASD and tape data sets may be used. The use and formats of these data sets vary from phase to phase.

● Table 2. Data Set Usage

Phase	Function	SYSUT1	SYSUT2	SYSUT3	SYSIN	SYSLIB
F1	Initial statement scan	Write source edited text			Read source text	
F2	Scan programmer macro definitions & subset each local dictionary	Write source text	Write/read local dictionary segments	Write edited text/macro dictionary	Read source text	Read source text (optional)
	Scan main portion of program	Write source edited text	Write/read local dictionary segments		Read source text	Read source text (optional)
	Scan system macro definitions; subset each local dictionary		Write/read local dictionary segments	Write edited text/macro dictionary		Read source text (optional)
	Subset global dictionary and local dictionary for main program		Read local dictionary segments main program			
F3	Conditional assembly/ macro generation	Read source edited text	Write source edited text	Read edited text/macro dictionaries		

Phase	Function	SYSUT1	SYSUT2	SYSUT3	SYSPRINT	SYSPUNCH/ SYSGO	SYSTERM
F7	Main line control	Write text*	Read text*	Write literal base table			
	Process symbols and literals	Write literals in text		Write symbol table segments			
	Process PUNCH statements & TESTRAN symbol table					Write PUNCH statements and TESTRAN symbol table	
	Process external symbol dictionary and cross-reference entries			Write external symbol dictionary segments & cross references			
F1	Process external symbol dictionary	Not used*	Write literal adjustment table*	Read literal base table and external symbol dictionary segments	Write external symbol dictionary	Write external symbol dictionary	
F8	Final assembly	Read text*	Read literal base table*	Write relocation dictionary, diagnostics, and flagged statements	Write text	Write object text	
	Process relocation dictionary entries			Write relocation dictionary			
FPP	Process relocation dictionaries			Read relocation dictionary		Write relocation dictionary	
	Process cross-reference list			Read cross-references	Write cross-reference list		
	Process assembly statistics			Read diagnostics	Write diagnostics		Write diagnostic information

*NOTE: If an odd number of iterations occur due to symbol table overflow in F7, the functions of SYSUT1 and SYSUT2 are reversed where indicated.



- Figure 2. Data Sets Used by the Assembler

SYSIN. This is a DASD, tape, or card reader resident data set containing the source text to be assembled. The source text consists of control text, programmer macro definitions, and text of the main portion of the program. The format is assumed to be 80-byte logical record card images, blocked as necessary.

SYSLIB. This is a DASD resident, partitioned data set containing system macro definitions and source text which may be COPYed into programmer macro definitions, into the main text of the program, or into system macro definitions. This may be blocked.

SYSPRINT. This is a printer, DASD, remote terminal, or magnetic tape resident data set containing output text for printing. This may be blocked.

SYSPUNCH. This data set contains the output text for punching. It may be punch, DASD, or magnetic tape resident. This may be blocked.

SYSGO. This is a DASD or magnetic tape resident data set containing the same output text as SYSPUNCH, but is used as input for the linkage editor. This may be blocked.

SYSTERM. This is a remote terminal resident data set containing diagnostic information. This may be blocked.

Control Program Services

Several control program services (system macro instructions) are used in conjunction with data and storage manipulations and program control transfers during the seven phases of the assembler. These macros are briefly described in Appendix B, and detailed information is given in the Supervisor and Data Management Macro Instructions publication. The macros used are DCB, GET, PUT, READ, WRITE, CHECK, NOTE, POINT, GETMAIN, FREEMAIN, FIND, OPEN, CLOSE, CLOSE (Type T=TCLOSE), LINK, XCTL, and RETURN.

Data management services are given in Table 3. System overhead estimates are given in Appendix C.

ORGANIZATION OF THE ASSEMBLER

Major Components

The assembler consists of the modules listed in Table 4.

Basic Functions

The assembler has two major functions in the processing of source programs: generation and assembly. See Figure 3.

The function of the generator portion is conditional assembly and expansion of macro instructions to one-for-one statements. Phases F1, F2, and F3 comprises the generator portion.

The functions of the assembly portion is to convert the one-for-one source statement and expanded macro instructions into machine language instructions and constants and to produce a relocatable object program. These functions are performed by Phases F7, FI, F8, and FPP. (There are no F4, F5, or F6 phases.)

ASM is the master root segment for calling routines and initialization procedures.

MAC contains and initializes the following I/O control program routines for Phases F1, F2, and F3.

- READ, WRITE, CHECK, NOTE, and POINT for the three work files.
- GET for SYSIN.

MAC is LINKed for ASM. See Flowchart 1. MAC in turn LINKs to Phase F1. MAC remains resident during Phase F1, F2, and F3. It is later overlaid by RTA.

Control is transferred from MAC to RTA.

RTA contains and initializes the control table for Phases F7, FI, F8, and FPP. RTA then LINKs to Phase F7 and remains resident during Phases F7, FI, F8, and FPP. Upon completion of the final assembler phase (Phase FPP), control is returned to RTA. RTA returns control to ASM.

There are two error abort phases -- F3E and ERR.

●Table 3. Use of Data Management Services

Phase	Data Sets								
	SYSUT 1	SYSUT 2	SYSUT3	SYSIN	SYSPRINT	SYSPUNCH	SYSGO	SYSLIB	SYSTERM
F1	OPEN WRITE CHECK	OPEN	OPEN	OPEN GET				OPEN	
F2	WRITE CHECK TCLOSE	READ WRITE NOTE POINT CHECK TCLOSE	WRITE NOTE POINT CHECK TCLOSE	GET CLOSE				READ NOTE POINT CHECK FIND CLOSE	
F3	READ CHECK TCLOSE NOTE POINT	WRITE CHECK TCLOSE	READ CHECK TCLOSE NOTE POINT						
F7	READ WRITE CHECK TCLOSE	READ WRITE CHECK TCLOSE	READ WRITE NOTE POINT CHECK			OPEN PUT	OPEN PUT		
FI	WRITE TCLOSE CHECK	WRITE TCLOSE CHECK	READ NOTE POINT CHECK		OPEN PUT	PUT	PUT		
F8	READ CHECK TCLOSE	READ CHECK TCLOSE	READ WRITE CHECK NOTE POINT		PUT	PUT	PUT		
FPP	READ WRITE CHECK TCLOSE CLOSE	READ WRITE CHECK TCLOSE CLOSE	READ CHECK NOTE POINT CLOSE		PUT CLOSE	PUT CLOSE	PUT CLOSE		OPEN PUT CLOSE
ERR	CLOSE	CLOSE	CLOSE		OPEN CLOSE PUT	CLOSE	CLOSE		

• Table 4. Organization of the Assembler

Phase	Module	Description
ASM	IEUASM	Master root segment
MAC	IEUMAC	Macro generator I/O package
F1	IEUF1	Macro generator initialization
F2	IEUF2 IEUF2A	Macro generator syntactical scan Macro generator dictionary routines
F3	IEUF3	Macro generator generation
F3E	IEUF3E	Phase F3 substitute for abort conditions
RTA	IEURTA IEUF7S* IEUF7L*	Root segment A phase Symbol table routine Log error routine
F7	IEUF7I IEUF7C IEUF7X IEUF7N IEUF7E IEUF7D IEUF7V IEUF7G	Initialization and I/O Main line control GET statement TESTRAN routine External symbol dictionary routine DC evaluation Expression evaluation routine DC GET routine
FI	IEUFI	Interlude phase
F8	IEUF8I IEUF8C IEUF8M IEUF8A IEUF8P IEUF8D IEUF8N IEUF8V IEUF8S IEUF8L	Initilization and I/O Main line control Machine operation processor Assembler operation processor Print routine DC evaluation Decimal/floating point conversion Expression evaluation routine Symbol table routine Log error routine
FPP	IEUFPP IEUFD	Post processor Diagnostic list
ERR	IEUERR	Permanent I/O error abort phase

* A routine that logically belongs to Phase F7, but resides in the module IEURTA.

Macro Generation and Conditional Assembly Phases

The macro generation and conditional assembly portion of the assembler requires three phases (F1, F2, and F3) and two passes over the source program text.

The function of the first pass (in Phase F2) is to scan the source text, determine the syntax of each statement, and produce edited text suitable for actual macro generation and conditional assembly.

Variable symbols, sequence symbols, macro names, and symbols appearing in the main portion of the program are collected and placed in either a global dictionary or in one of the local dictionaries, depending on the type of the symbol and the context. Extraneous information is removed from these dictionaries, and a subsetted dictionary is produced for the main portion of the program, for each system and programmer macro used in it, and for global information.

The second pass (in Phase F3) performs the actual macro generation and conditional assembly using the edited source text and subsetted dictionary information created during the first pass. A one-for-one edited text version is produced for input to the subsequent assembly phases.

Phase F1 (Initialization and Assignment). Phase F1 performs the primary initialization for the macro generation and assembly phases. The operating environment is established by obtaining data sets from the control program for utility, input, and library functions, and by obtaining main storage for tables, etc.

Phase F2 (Statement Scan). Phase F2 scans for programmer macro definitions, the main program, and system macro definitions and produces edited text for the main portion of the program and for each macro definition for input to Phase F3. Phase F2 creates the global and local dictionaries and prepares subsetted versions of them for Phase F3. Syntactic errors are detected and flagged for subsequent error processing. If an abort condition arises, Phase F2 passes control to Phase F3E, a substitute version of Phase F3. An abort condition can be caused by any one of the following:

- The global dictionary fills up.
- The local dictionary exceeds 64K bytes on the overflow file or in core.
- The subsetting area is too small.

Phase F3 (Conditional Assembly and Macro Generation). Using the edited text and dictionaries produced in Phase F2, Phase F3 generates one-for-one statements in edited text form from the macro definitions and performs conditional assembly. This output serves as input to Phase F7, the first of the assembly phases.

Assembly Phases

Phase F7 (Initial Assembly). Phase F7 processes symbols and literals, builds the symbol table and the external symbol dictionary, and assigns relative locations to all statements in the text. The symbol table contains the definition (name and attributes) of every declared symbol. Sharing the same storage area, similar information is col-

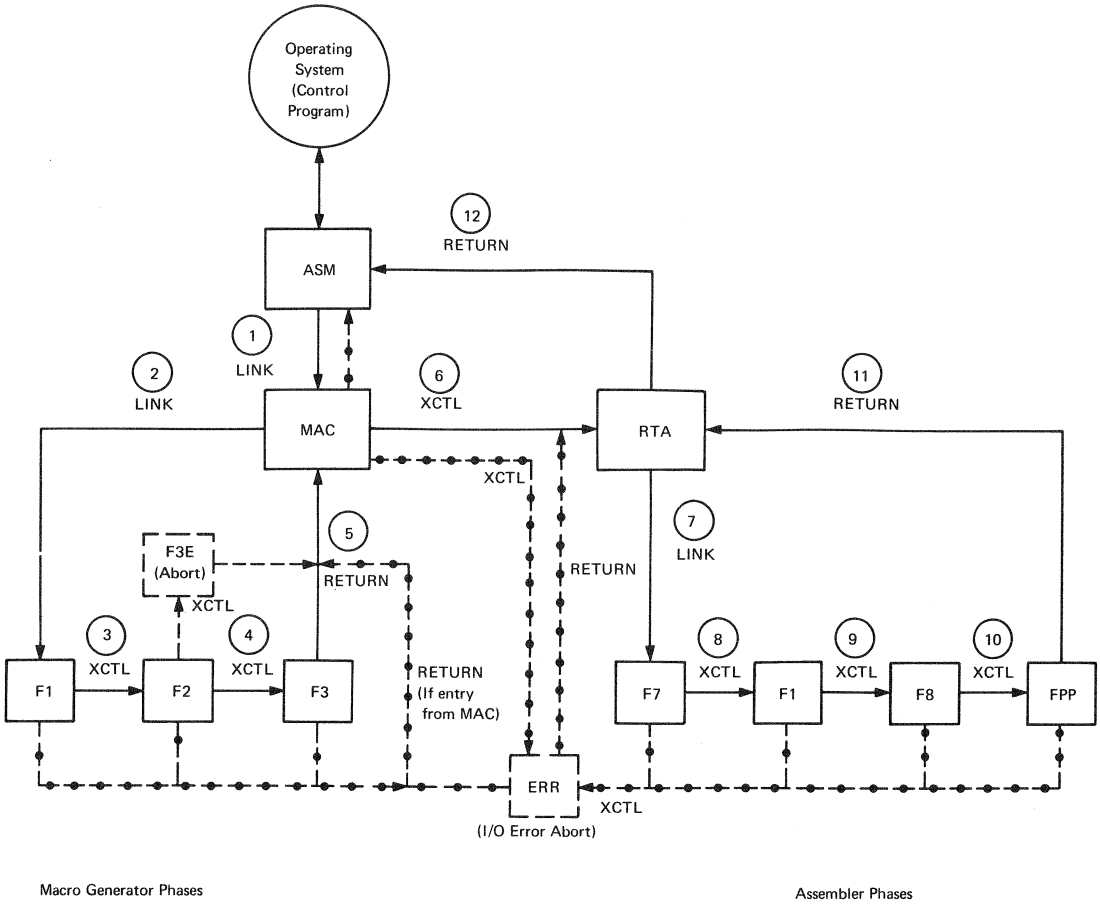


Figure 3. Program Flow

lected for literals. Only one symbol table is constructed if the allocated core can accommodate the entire table. If necessary, the external symbol dictionary, except for two segments, will be stored on SYSUT3 to make room in core for the symbol table. If there is still not enough core, construction of the symbol table is suspended at the overflow point, and all input statements are processed with data from the table (when applicable). This symbol table is then discarded and a new symbol table is built with symbols and literals that were not included in the eliminated table. This procedure is repeated as long as overflow conditions arise. Each completed symbol table is fully utilized and eliminated before the next one is built. Expressions appearing in all statements which require previous definition are evaluated in the process of location assignment, and a table of cross-reference entries is built for all symbols which are defined and/or referenced.

Phase FI (Interlude). Phase FI writes the external symbol dictionary on data sets SYSPRINT and SYSPUNCH and/or SYSGO, and constructs the literal adjustment table for use in Phase F8. If the external symbol dictionary was written on SYSUT3 during Phase F7 because of excessive core occupancy by the symbol table, the entire external symbol dictionary is read back into main storage.

Phase F8 (Final Assembly). Phase F8 completes the symbol processing on all operands using the last (or only) symbol table created in Phase F7. USING tables and the relocation dictionary are built, all address expressions are evaluated, and the operand addresses are converted to base-displacement format. Finally, the assembled text is written in relocatable object program format on SYSPUNCH and/or SYSGO and in program listing format on SYSPRINT.

Phase FPP (Post Processor). Phase FPP formats and writes the relocation dictionary table. If requested, the cross-reference entries are sorted, and the cross-reference list is prepared and written. Diagnostic messages and statistics are prepared and written on SYSPRINT, and diagnostic information is prepared and written on SYSTERM. Phase FPP also writes the END card.

PROGRAM LEVELS

In order to establish a uniform method of communication and to control flow between the phases and phase routines, the assembler is structured on three levels:

- Level 0 - Phase initialization, storage allocation, mainline control, phase call, and Operating System interface.
- Level 1 - Functional routines.
- Level 2 - Common subroutines and common tables.

Functional routines are assembled as independent control sections with a single entry point and a return to mainline control. Functional routines do not call other level 1 routines. Any communication required between functional routines is through mainline control and the common table area.

Common subroutines are also assembled as independent control sections. Linkage between functional routines and common subroutines as described in Linkage Conventions below.

ASSEMBLER CONTROL TABLE

Linkage between mainline control, functional routine, common subroutines, and common tables is accomplished through the use of the assembler control table. The location of the assembler control table is always available to all levels of routines and subroutines through the general purpose register ACT.

The assembler control table is divided into five parts:

1. Mainline control/functional routine linkage/return algorithms.
2. Functional routine pointers.
3. Common subroutine pointers.
4. Common table pointers.
5. Switches

Each of these five sections is in a fixed location with respect to the ACT pointer, and each element in a given section is in a fixed location with respect to the start of that section. A central dimensioning deck of EQU cards is used by each control section to reference the assembler control table symbolically.

Example:

```
ACT    EQU    (General register number for
              assembler control table
              pointer)
CT1    EQU    0
CT2    EQU    Relative location of section
              2
CT3    EQU    Relative location of section
              3
CT4    EQU    Relative location of section
              4
CT5    EQU    Relative location of section
              5
```

GENERAL REGISTER ASSIGNMENTS

General registers 3-15 are referenced symbolically by all assembler subprograms. The equate cards are described in Table 5 and in the paragraphs below. They are included in all control section decks to provide the initial assignments for general register symbols.

ACT (Assembler Control Table) Pointer

This general register contains the absolute starting location for the assembler control table. This register is set by the control routine during phase initialization and remains set for the duration of the phase. ACT should be considered as a read-only register for the use of all routines and subroutines and as a base register for assembler control table references.

FRB (Functional Routine Base Register)

This general register contains the base address for the current functional routine. FRB is set by the control routine and is used to link with functional routines. The USING statement for each functional routine uses FRB as the second operand and the name of the functional routine's entry point as the first operand.

Example:

```
USING    BEGIN,FRB
```

FRB should not be used by common subroutines or by functional routines other than in their USING statements. Functional routines

Table 5. General Register Assignments

Symbol	Tentative EQU	Purpose	Restrictions	Remarks
GR0 GR1, GR2	Absolute		None	
ACT	3	Assembler control table pointer	Read only register	
SRB	8	Common subroutine base register	Set by calling routine. Used by S/R in USING statement	Four contiguous registers beginning with an even register
SRR	9	Common subroutine return register	Set by calling routine with BALR, SRR, SRB	
SP1	10	Common subroutine parameter 1	Used to transfer parameters between calling routine and common subroutines	
SP2	11	Common subroutine parameter 2		
GRX	14	General purpose registers X, Y, and Z, respectively	No restrictions. Completely volatile.	GRX and GRY are two contiguous registers beginning with an even register. GRZ is unpredictable.
GRY	15			
GRZ	13			
GRA	4	General purpose register A, B, C, and D, respectively.	Used by functional routines. Cannot be changed by common subroutines.	Four contiguous registers beginning with an even register
GRB	5			
GRC	6			
GRD	7			
FRB	12	Functional routine base register	Set by control routine. Used by functional routines in USING statement.	
CRB*	GRC	Control routine base register	Saved/restored by control routine when control is passed to functional routine	(Sample applications)
CRR*	GRD	Control routine return address		

*CRB and CRR are applications of the usage of general purpose registers for specific assignments.

do not link directly with other functional routines. Therefore, FRB remains intact during the operation of a functional routine.

SRB* (Subroutine Base Address)

This general register contains the base address for the current common subroutine. SRB is set by the calling routine and is used to link with common subroutines. The USING statement for each common subroutine uses SRB as the second operand and the name of the subroutine's entry point as the first operand.

Example:

```
USING    SRENTR,SRB
```

SRR* (Subroutine Return Address)

This general register contains the address through which the current common subroutine

*SRB, SRR, SP1, and SP2 may be used as general purpose registers by level 0 and level 1 routines if care is taken not to conflict with subroutine calls. If a subroutine calls another subroutine, it is the responsibility of the first subroutine to restore SRB and SRR.

returns to the calling routine. SRR is set by the calling routine with a BALR SRR,SRB.

SP1*, SP2* (Subroutine Parameters)

These general registers are used to transfer parameter(s) between the calling routine and the called subroutine. SP1 and SP2 are two contiguous registers beginning with an even register. If a parameter list exceeds the combined length of SP1 and SP2, SP1 is used as a pointer to the parameter list storage location.

GRX, GRY, GRZ (General Purpose Registers)

There are no restrictions on the use of these general registers. They may be used by any level routine and are not saved/restored by called subroutines and operating system functions.

GRA, GRB, GRC, GRD (Functional Routine General Purpose Registers)

These are four contiguous registers beginning with an even numbered register. Some subroutines could use these registers and restore them to their original value before returning to the calling routine. The control routine is required to save and restore these registers for its own use when transferring control to a functional routine.

LINKAGE CONVENTIONS

Subroutine Linkage

Common subroutines are linked to the mainline control and functional routines through the following procedure:

- The calling routine loads parameters into SP1 and SP2. If the parameter list exceeds two words in length, a pointer is used in SP1 or SP2 to point to the storage location of the parameter list.
- The calling routine loads the address of the subroutines in SRB. The subroutine address constants (ADCONs) are located in the assembler control table.

  Example:

```
L     SRB,SUBAD(ACT)
```

- The calling routine transfers control to the address specified by SRB, storing the return address into SRR.

  Example:

```
BALR     SRR,SRB
```

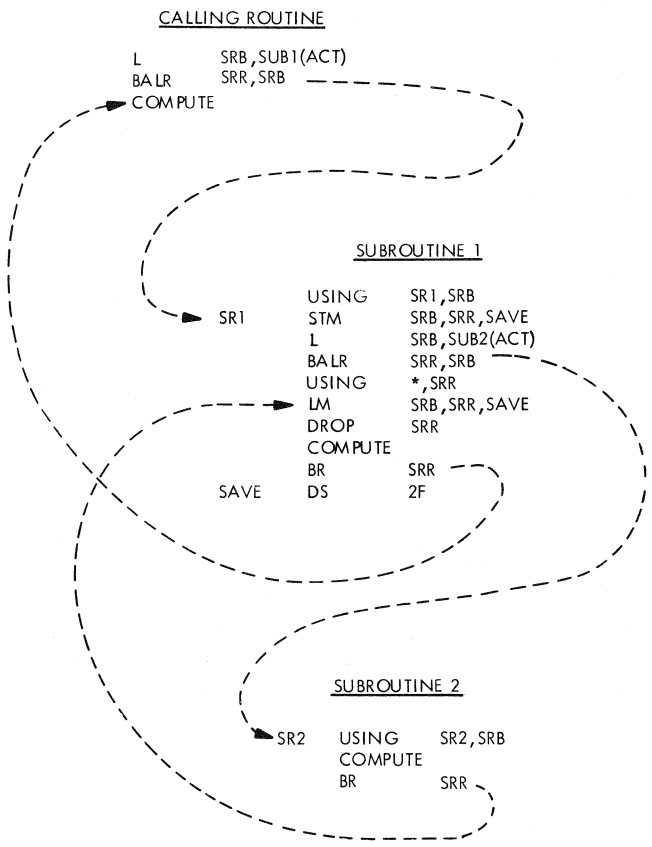
- The called subroutine returns control to the calling routine by branching to the address specified by SRR.

  Example:

```
BR     SRR
```

NOTE: If a common subroutine calls another common subroutine, the same procedure will be used. Therefore, it is necessary for the first subroutine to save and restore its own base and return registers.

Example:



Common Table Linkage

All tables and data areas used by more than one routine or subroutine are considered a

common table and follow the rules defined herein.

The starting location of each common table is contained in the assembler control table. Therefore, any reference to common control tables is preceded by loading the starting address for that table into a general register from the assembler control table.

Fields within common tables are referenced symbolically.

Example:

The starting location for TABLE is contained in ACT+56. TABLE consists of N entries. Each entry consists of three fields: FIELDA (2 bytes), FIELDB (1 byte), and FIELDC (5 bytes).

The dimensioning deck required to reference TABLE is as follows:

```
TABLE   EQU   56   (Rel. loc. of TABLE
                    pointer in ACT)
FIELDA  EQU    0   (Rel. loc. of FIELDA
                    in TABLE entry)
FIELDB  EQU    2   (Rel. loc. of FIELDB
                    in TABLE entry)
FIELDC  EQU    3   (Rel. loc. of FIELDC
                    in TABLE entry)
```

In the following sample coding, X is the register containing the TABLE pointer.

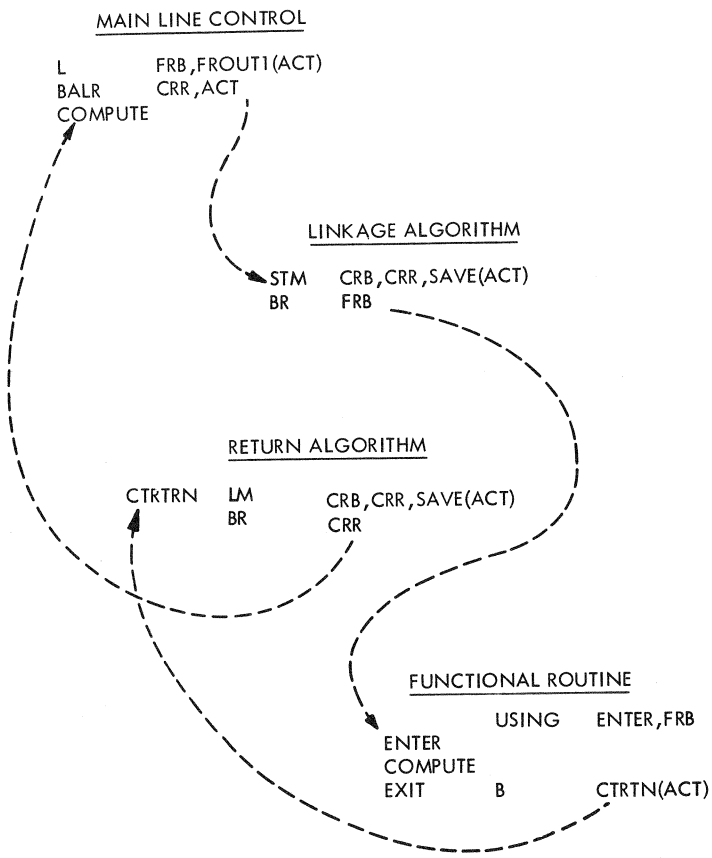
```
L       X,TABLE(ACT)
MVC     FIELDA(2,X),TEMP
MVC     FIELDB(1,X),FIELDA+1(X)
```

Linkage Between Mainline Control and Functional Routines

The mainline control routine transfers control to the functional routines by loading the functional routine base address into FRB from a table of pointers in the assembler control table. An algorithm in the assembler control table then saves the mainline control base and return registers, and branches to the functional routine pointed to by FRB.

The functional routine returns control to mainline control by branching to the return algorithm in the assembler control table. The return algorithm restores the mainline control base and return registers, and returns to mainline control.

Example:

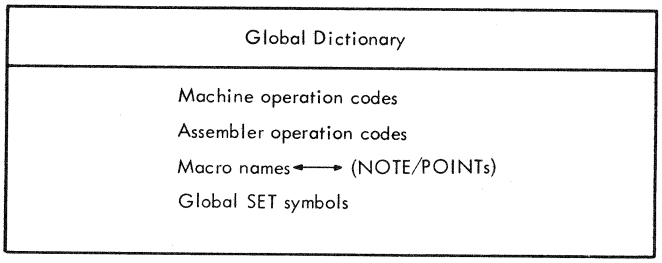


GENERAL

The assembler builds dictionaries and tables for its own use. Special facilities are used to enter new records and to access already stored records for adding or retrieving data in these designated areas. Hash tables, hashing algorithm, and chaining are the tools employed to accomplish the required data manipulations in dictionaries and certain tables.

DICTIONARY TYPES AND STRUCTURES

The assembler uses two types of dictionaries: a global (permanent) dictionary and a local (transient) dictionary. See Figure 4.



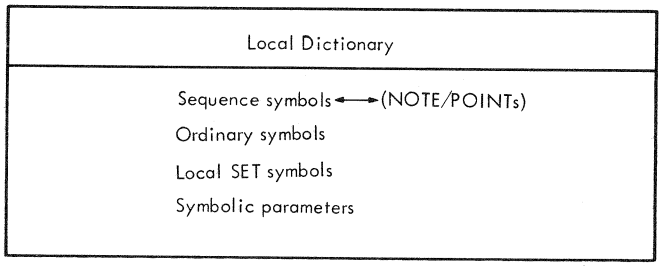


Figure 4. Dictionary Structure

There is only one global dictionary. It is core resident, and it contains all machine and assembler operation codes, all macro names, and all global SET variables.

There are many local dictionaries, one for each macro definition and one for open code (main text). The local dictionaries contain sequence symbols, ordinary symbols, local SET symbols, and macro instruction parameters. Local dictionaries contain items needed for insertion into edited text and for macro generation during Phase F3, and can overflow onto a utility data set during construction.

Each local macro dictionary is subsetted at the end of the processing of the macro definition (Phase F2). The subsetted dictionary is written on the system utility data set immediately following the edited text format of the macro definition.

The global and the open code local dictionaries are subsetted at the end of Phase F2. Subsetting removes flag bytes, symbolic names, big "A" pointers, and little "a" pointers. The global and main text local dictionaries are placed in core and remain there throughout Phase F3.

SYMBOL TABLE STRUCTURE

The symbol table is a collection of the attributes of symbols and literals. It is actually composed of two tables, one for symbols and one for literals; however, the same physical storage area is used for both and the two different types of entries are intermingled in storage.

The symbol table is a compact means to rapidly obtain the attributes of a given symbol or literal. The techniques used to store and retrieve symbol table information are hashing and chaining.

The symbol table is always in core and it consists of the symbol table proper and two hash tables, one for symbols and one for literals. The external symbol dictionary occupies the high-numbered portion of the alloted storage, but is output on a data set if the space is needed by the symbol table.

The symbol hash entries are three-byte addresses whose locations are obtained from the symbol names. Different names may have the same hash entry. If so, the entry points to the last synonym entered, which has a chain pointer to the preceding one. Literal entries are referenced through a literal hash table which contains a three-byte pointer for each of four literal strings. There is a literal string for each of the literal lengths of 1, 2, 4, and 8 bytes. Each literal hash pointer points to the first entry in the proper string. The entries in each string are chained together in a forward manner. (See Hash Table and Chaining, below.)

HASH TABLE

A hash table is used by the assembler for inserting or locating variable or fixed-length record entries in dictionaries and symbol tables. A hash table consists of fixed-length address entries (called point-

ers) 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 space reserved for the table. When it is desired to make an entry in the dictionary/table, e.g., enter a global symbol declaration, or to locate an entry in the dictionary/table, e.g., to obtain the relative address of a symbol, the associated symbol or other datum must first be randomized to produce an index number. This is called hashing. (Operation codes are included in the generation of index numbers for the macro dictionaries.) 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 record entry, randomizing to this index number, has been entered in the dictionary/table. Records are entered in the dictionary/table sequentially, and a dictionary/table pointer, containing the next available address, is used for inserting new records. Several different data (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.

CHAINING

Chaining is a technique whereby an entry to one record points to the next record, and so on. Forward chaining and backward chaining are the two types of chaining used by the assembler.

In forward chaining, a hash table pointer entry points to the first entry of a chain. The first field of each entry contains a chaining address pointing to the next entry in the chain. The last entry in each chain has all zeros in the chaining address field.

In backward chaining, a hash table pointer entry points to the last entry of a chain. The first field of each entry contains a chaining address pointing to the preceding entry in the chain. The first entry in each chain has all zeros in the chaining address field, or, in certain applications, the pointer field is eliminated in the first entry.

Forward Chaining Techniques

The symbol, literal, or other datum 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 will be zero if no other item has yet hashed to the same index number, i.e., this is the first record entry for this index number. If this is not the first entry to this index number, the hash table will contain the address of the first record entered in this chain. The record at that address will be checked for duplication. If there is no duplication, the content of the chain pointer field is checked in the record. This pointer will be either a chaining address pointing to the location of the next record in the chain, or zero. Zero indicates that this is the last (or only) record in the chain. If the pointer field contains a chaining address, the next record is checked for duplication. Again, if there is no duplication, it is checked for a zero chain pointer (zero=last record in the chain). The scan is continued in this manner until a duplication is found, when the precedure is terminated without making a new entry, or until a zero pointer is reached, in which case the new record is entered in the dictionary/table. In the latter case, the zero pointer is replaced with the address of the dictionary/table pointer, i.e., the address of the next available dictionary/table location, the new record is entered at this location with a zero pointer, and the dictionary/table pointer is updated with the length of the current entry.

The procedure used to locate records in the dictionaries/tables is the same as entering, except that when the compared records are equal, the pertinent information is extracted, or the value information is inserted, as the case may be. See Figure 5.

Backward Chaining Techniques

The record 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 will be zero if no other record has yet hashed to this index number. If this is not the first entry, the hash table will contain the address of the last record entered, i.e., the most current entry. The record at this address will be checked for duplication. If there is no duplication, the content of the record's chain pointer field is checked. This chain pointer field contains the address of the previous entry in the chain, or zero, if it is the first (or only) entry in the chain.

The chain is scanned starting with the last entry and continuing through the first entry, or until a duplicate record is encountered. In the latter case, the scan is terminated and the record is not entered. If there is no duplication, the address of the last record in the chain is placed in the chaining address field of the record entered. The address of the dictionary/table pointer, i.e., the address of the next available location in the dictionary/table area, replaces the pointer

in the hash table, and the record is inserted at this address in the dictionary/table.

The dictionary/table pointer is updated by the length of the record just stored to indicate the new available storage address.

The procedure to locate records in the dictionaries/tables is the same as entering, except that when the compared records are equal, the pertinent information is extracted, or the value information is inserted, as the case may be. See Figure 6.

Chaining Usage

In Phase F2, forward chaining is used in building the global dictionary. However, in addition to the many forward chains created, all macro name entries in the global dictionary are linked together by backward chaining. Therefore, each macro entry has two pointer fields. The first field points forward to the next record in the chain, which originated from the same hash table pointer, and the last field points backward to the

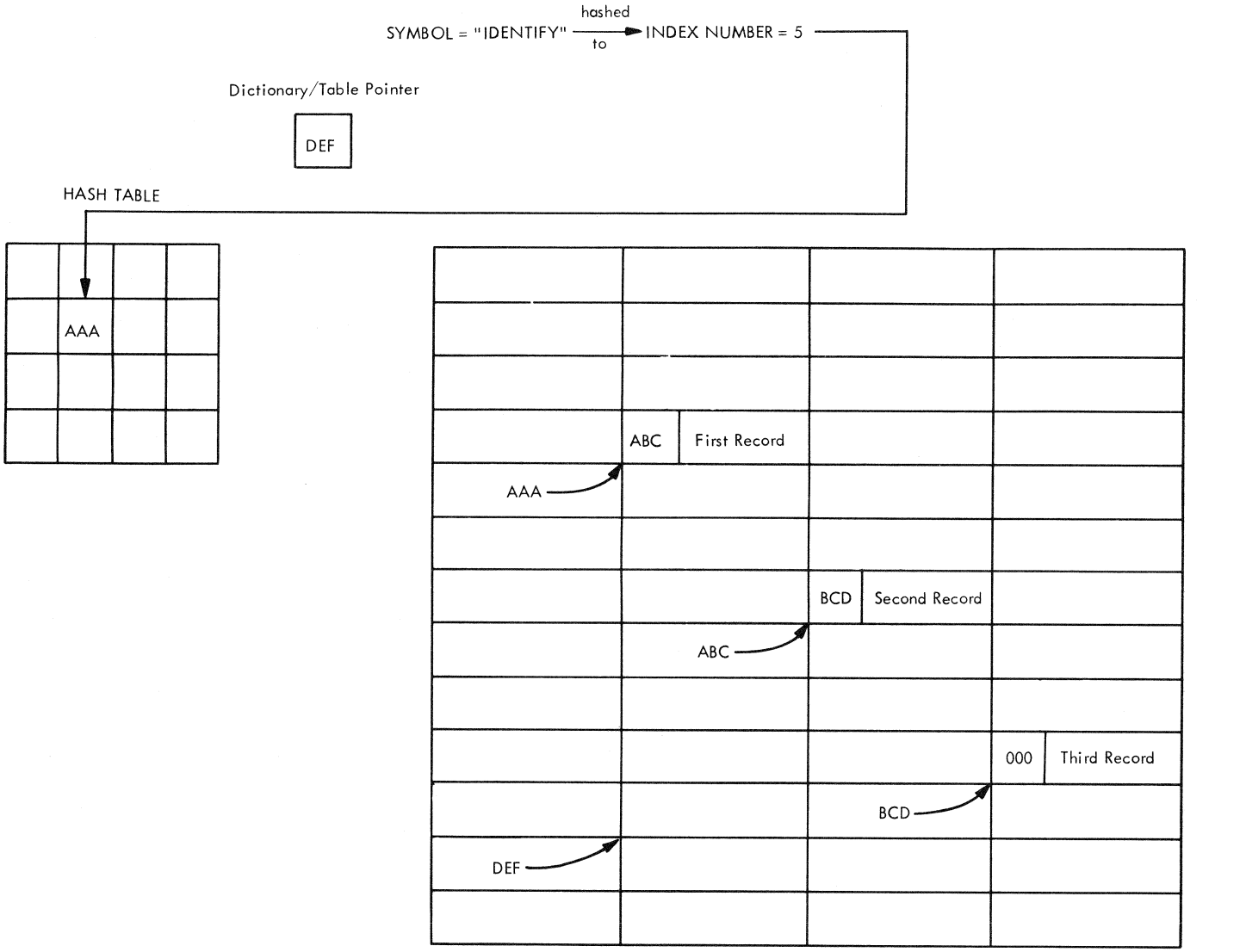


Figure 5. Hash Table and Forward Chaining

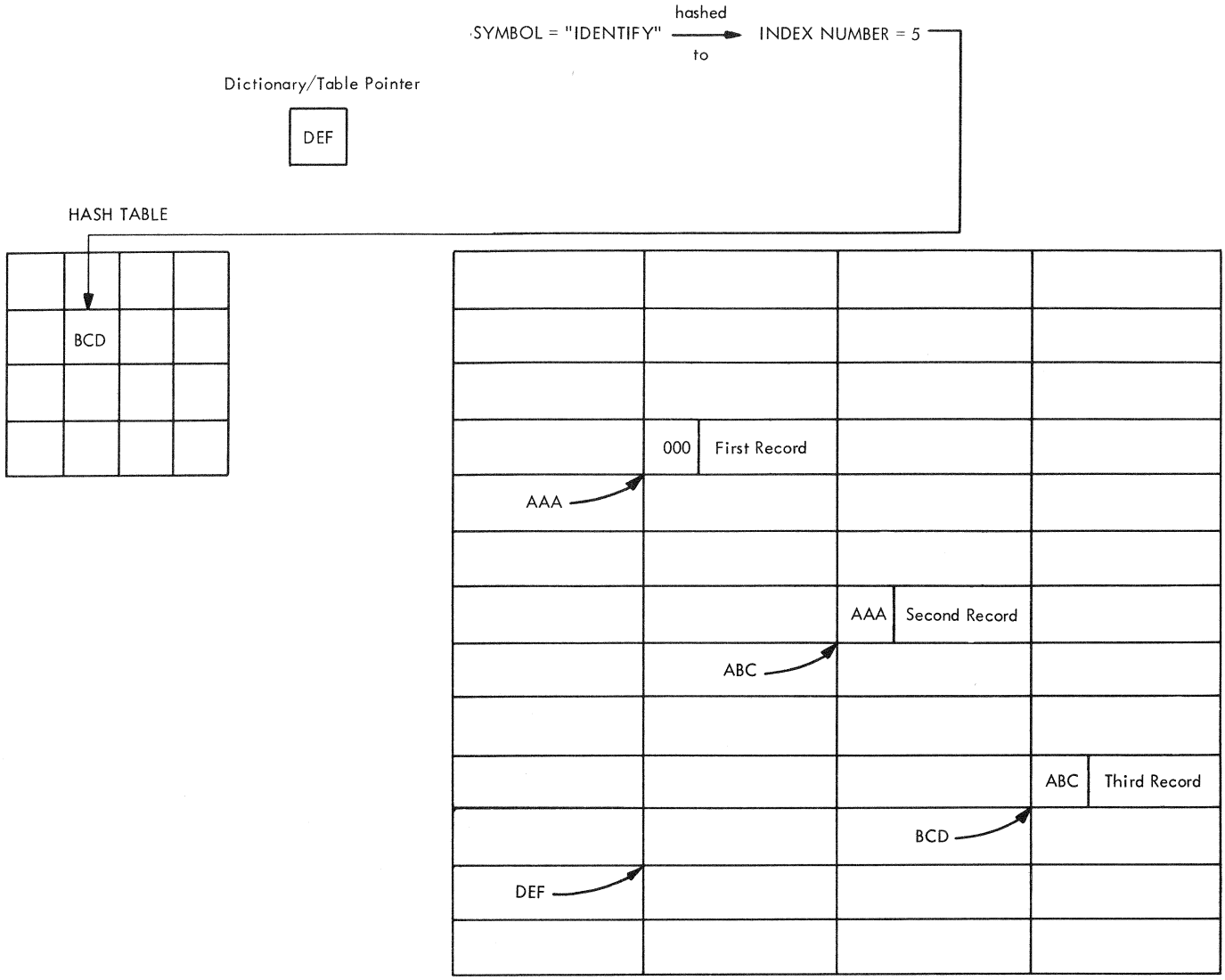


Figure 6. Hash Table and Backward Chaining

preceding macro entry in the macro chain. The first macro entry in the macro chain has a zero macro chain pointer.

Backward chaining is also used in Phase F2 to build local dictionaries.

In Phases F7 and F8, the symbol table area is shared by symbols and literals in a random fashion. Symbol entries are reached through pointers located in the symbol hash table and are chained backwards. The first symbol entry has no pointer field. Literal entries are reached through the literal hash table and are chained forward.

OVERALL OPERATION (FLOWCHART 2)

Phase F1 performs initialization for Phase F2. Phase F1 does a GETMAIN for needed common area, and from this determines an optimum buffer size. Phase F1 generates a hash table and associated global dictionary. Machine operation codes and assembler operation codes are inserted into the global dictionary at this time.

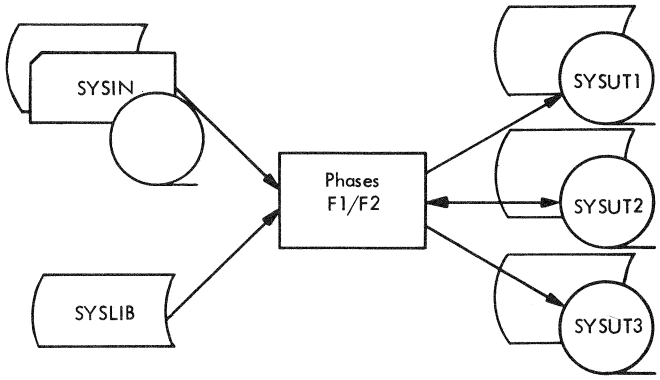


Figure 7. I/O Flow for Phases F1 and F2

FUNCTIONS

The first five data sets, SYSIN, SYSLIB, SYSUT1, SYSUT2, and SYSUT3, are opened. See Figure 7. The assignments of the three utility data sets within the assembler are influenced by the channel configuration and device types. The parameter list on the EXEC card is scanned and processed for reference by subsequent phases. Any errors are diagnosed.

Phase F1 reads the first card and, if that card is not an ICTL or OPSYN card, it XCTLs to Phase F2. If it encounters an ICTL or OPSYN card, it first processes all ICTL and OPSYN cards. When it reads a record that is not an ICTL or OPSYN instruction, it transfers control to Phase F2.

SUBROUTINES

ICTL

Sets the input control values as specified in the ICTL instruction.

OPSYN

This routine processes OPSYN definitions. If the symbol in the name field is not an existing operation code (valid for OPSYN), the routine creates a global dictionary entry -- identical to the entry for the operation code in the operand field -- and hashes it on a chain for OPSYN-defined op codes. If the name is an existing op code, the routine changes the internal machine operation code and (if required) the type code and ASC to that of the op code in the operand field. If the op code is to be removed (blank operand field), the routine zeros out the first byte of the op code name in its global dictionary entry.

This phase performs all necessary functions to enable conditional assembly by the next phase, F3. It reads the input from SYSIN and performs a syntactical scan of all the input. See Figure 7. Any undefined operation codes are assumed to be system macro instructions. (If the DOS assembler option is specified, F2 treats CXD, DXD, and OPSYN instructions as undefined. It also treats L-type and Q-type DC and DS instructions and extended precision machine instructions as undefined.) A search of the system library is made and the system macros are edited just as are programmer macros. In addition to scanning, Phase F2 inserts various items into appropriate dictionaries.

Items present in the dictionaries are needed for insertion into edited text and generation by Phase F3.

When a dictionary is complete, it is subsetted and all items except those needed by Phase F3 are deleted.

FUNCTIONS

Phase F2 is divided into the following four logical sections:

- Programmer macro definition scan and dictionary build.
- Main text scan and dictionary build.
- System macro definition scan and dictionary build.
- Subsetting of the global dictionary and all local dictionaries.

Because these sections have many overlapping functions, they are contained in a single phase.

Programmer Macro Definition Scan and Dictionary Build

Source text is read from SYSIN and optionally from SYSLIB (using FIND) if COPY assembler instructions are encountered in the macro definitions. A copy of the source text is written on SYSUT1 for later inclusion in the SYSPRINT data set, while edited text is written on SYSUT3.

A chaining technique is used to make entries in the dictionaries. (See Chaining Techniques in the Dictionary and Table Construction Techniques section.) As the source text is scanned, entries are made in the global dictionary for macro mnemonics and global SET symbols. Entries are made in the local dictionary corresponding to the given macro for sequence symbols, parameters, and local variable symbols. Segments of the given local dictionary are written on SYSUT2, and the macro definition in an edited text format is written on SYSUT3. When the end of the given macro definition is encountered, the local dictionary for that macro is subsetted and written on SYSUT3 immediately following the corresponding edited text.

Main Text Scan and Dictionary Build

Source text is read from SYSIN and optionally from SYSLIB (using FIND) if COPY assembler instructions are present. The source text of the main portion of the program is combined with an edited form of the main text and written on SYSUT1. This text is blocked to minimize rotational delay timing problems inherent with DASD devices.

A hashing technique is used to enter information in the dictionaries. (See Hash Tables in the Dictionary and Table Construction Techniques Section.) The same global dictionary used in the programmer macro definition scan is used in the main text scan to contain macro names, OPSYN definitions and global SET variables. The local dictionary for the main text is used to store main text symbols as well as local variable symbols and sequence symbols. Segments of this local dictionary are written on SYSUT2. A circular buffer pool is employed with a backward chaining technique to keep as many of the most recently written local dictionary segments in main storage as possible. However, for large programs it may be necessary to read back one of the segments no longer remaining in main storage. Further, the boundary between the global dictionary and the current local dictionary may be adjusted to permit a larger number of global dictionary entries, thereby reducing the main storage for the local dictionary and possibly requiring more segments to be read back.

System Macro Definition Scan and Dictionary Build

The global mnemonics dictionary contains mnemonics of macros used in the programmer macro definitions and in the main text of the program. Mnemonics of macros which were not defined as programmer macros are chained together and considered to be system macros. One by one these macros are located in SYSLIB (using FIND), and their definitions are scanned and edited similar to the programmer macro definitions above. Text is read from SYSLIB. Since no listing of system macro definitions is required, only the edited text and subsetted local dictionaries are written (on SYSUT3).

Entries are made in the global dictionary for macro mnemonics and global SET symbols, and for each system macro definition a local dictionary is created in which sequence symbols, parameters, and local variable symbols are entered. As in the case of programmer macro definitions, each local dictionary is subsetted to form a macro dictionary, which is written on SYSUT3 immediately following the edited text for the macro.

This process continues until all system macro definitions have been processed or until no more macro names can be located in the library, at which time undefined macros are treated as illegal instructions.

Global Dictionary and Main Text Local Dictionary Subsetting

After editing all text preparatory to the actual conditional assembly and macro generation, the global dictionary is subsetted and placed in main storage. The main text local dictionary is then subsetted and also placed in main storage.

GLOBAL DICTIONARY ENTRY FORMATS

Defined Operation Codes

This format is as follows:

Bytes 2	1	1 to 8	1	1	1
"A" pointer	Flag	Mnemonic Operation Code	Internal Machine or Assembler Operation Code	R1 Mask	ASC

"A" Pointer (big A pointer) Relative forward chaining address of the next consecutive entry in a dictionary chain.

Flag -
- Bit 0 Not used
- Bits 1-4 0000 - Normal machine op
  0001 - Assembler op
  0010 - Extended mnemonics
- Bits 5-7 Length of operation code minus one (L-1)

R1 Mask - R1 field for extended mnemonics. The extension of op codes field is omitted for machine operation code entries. For two-byte op codes in System/370 instructions, the second half of this byte contains the displacement into the table TASTABLE. This table is internal to IEUF8M and contains the second byte of two-byte codes.

ASC - Assembler switch code. (See Edited Text Record Fixed Field Format in the "Phase F7" section.)

Macro Name Entry

This format is as follows:

Bytes 2	1	1 to 8	2	3	2
"A" pointer	Flag	Mnemonic	"a" pointer	NOTE/ POINT	MCP

"A" Pointer - Forward chaining address

Flag -
- Bit 0 0 - Normal global variables.
  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.

- Bits 1-4 0000 - Op codes
  0001 - Internal Assembler op codes
  0010 - Extended mnemonics
  0011 - Macro names
  0100 - Global A variables
  0101 - Global B variables
  0110 - Global C variables
- Bits 5-7 Length of BCD entry minus one (L-1)

Mnemonic - The macro name in byte format.

"a" Pointer - Little "a" pointer to a location in the Phase F3 dictionary that will contain the subsetted entry for this mnemonic.

NOTE/POINT - NOTE/POINT Address. Before the dictionary associated with this mnemonic is subsetted, this entry will contain the NOTE location of the beginning of the fully edited text for the corresponding macro definition. After the local dictionary has been subsetted, this field contains the NOTEd location of the subset dictionary, which will in turn contain the NOTEd position of the fully-edited text of the macro definition.

MCP - Macro chain pointer. The backwards chaining address of the preceding macro name entry in the dictionary.

Global SET Variable Symbol Entry

This format is as follows:

Bytes 2	1	2 to 8	2	2
"A" pointer	Flag	SET variable symbol	"a" pointer	C

"A" Pointer - Forward chaining address.

Flag - Same as in the macro name entry.

SET Variable Symbol - The SET variable symbol in byte format.

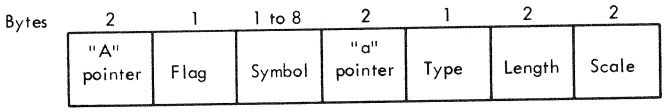
"a" Pointer - Same as in the macro name entry.

C - Dimension, the declared SET variable dimension. This will be zero if undimensioned.

LOCAL DICTIONARY ENTRY FORMATS

Open Code Ordinary Symbols

This format is as follows:



"A" Pointer (Big A pointer) - Backward chaining address of the preceding entry in a dictionary chain.

Flag -

Bit 0 — 0 - Synonym (part of a chain)
1 - End of the chain

Bits 1-4 —
1000 - Sequence symbols
1001 - Parameters
1010 - Ordinary Symbols
1100 - Local A variables
1101 - Local B variables
1110 - Local C variables

Bits 5-7 — Length of BCD entry minus one (L-1)

Symbol - The symbol in character format.

"a" Pointer - Same as in the macro name entry.

Type - Type attribute. See Table 6.

Length - Length attribute:

Scale - Scale attribute:

Bit 0 — 0 - Positive
1 - Negative

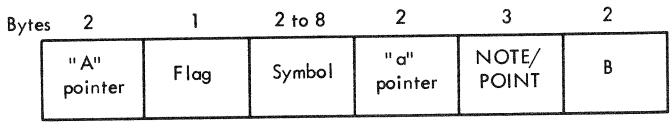
Bits 1-15 Scale attribute

Table 6. Type Indicators (Phases F2/F3)

Description of Record	Type Indicator
Source statement continuation record	08
Error record (warning message)	08
End of data set	0A
Error record (not warning message)	0D
Edited text records (machine instructions, DC, DS etc.)	00
CSECT, DSECT, START edited text record	01
AGO edited text record	02
AIF edited text record	03
SETx and ACTR edited text record	04
Macro instruction edited text record	05
Macro definition prototype statement edited text record	06
MEXIT and MEND edited text flag record	07
ANOP edited text flag record	09
Macro instruction or prototype operand value record	0B
End of macro instruction or prototype record	0C

Sequence Symbols

This format is as follows:



"A" Pointer - Backward chaining address.

Flag - Same as in the open code ordinary symbols.

Symbol - The symbol in character format.

"a" Pointer - Same as in the macro name entry.

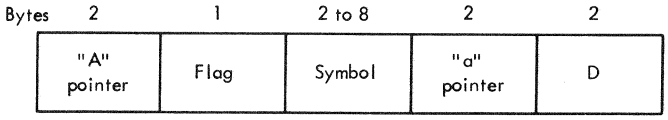
NOTE/POINT - The pointer to the block in which the fully-edited text for the statement named by the sequence symbol begins.

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

This format is as follows:



"A" Pointer - Backward chaining address.

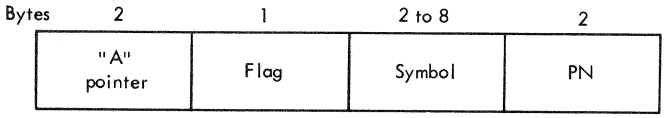
Flag - Same as in the open code ordinary symbols.

Symbol - The symbol in character format.

"a" Pointer - Same as in the macro name entry.

D - The declared dimension of the local SET variable symbol. It will be zero if the symbol is undimensioned.

Macro Prototype Symbolic Parameters. This format is as follows:



This format is for both keyword and positional type.

"A" Pointer - Backward chaining address.

Flag - Same as in the open code ordinary symbols.

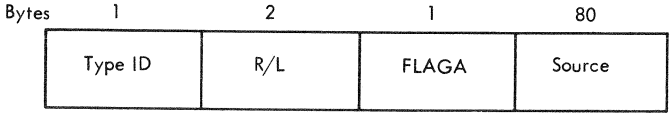
Symbol - The symbol in character format.

PN - The operand position number assigned to the symbolic parameter.

RECORD FORMATS

Source Record

This format is as follows:



Type ID - 08

R/L - Record length

FLAGA -

Bit 0 Not used.

Bits 1-3 Record type

- 000 - Print as is. Source record only
- 001 - Error record.
- 010 - Print as is, but do not display a statement number. Source record only.
- 011 - Print as is if GEN option is on. Steps statement number counter. Source record only
- 100 - Process only. Edited text and logical statements only.
- 101 - Internal assembler control record.
- 110 - Process this record and construct source record for print.
- 111 - Process this record and construct source record for print if GEN option is on.

Bit 4 Not used. (This bit is activated in Phase F7.)

Bit 5 Error record follows indicator.

Bit 6 Continuation card indicator.

Bit 7 Not used. (This bit is activated in Phase F7.)

Source - When created in F2 from source records from SYSIN or SYSLIB, source will be 80 bytes long.

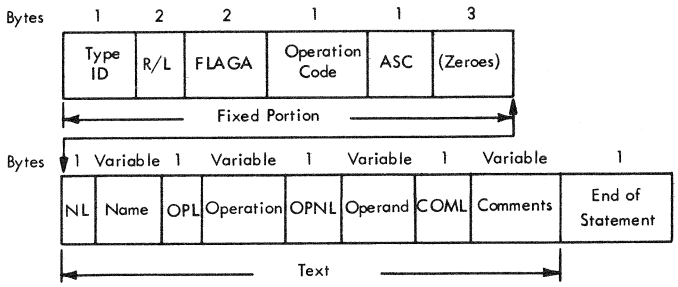
NOTE: If a source statement has a continuation card, an ERROR, and is split between buffers, then the order on SYSUT1 is as follows:

- Source record (first part).
- Error record.
- Source record (second part).
- Source continue record (from continuation card).

Source records are not created in Phase F2 for the system macro definition source statements (with the exception of comments for generation: "*").

Logical Statement

This format is as follows:



Type ID - See Table 6.

R/L - Record length.

FLAGA - Same as in the edited text record fixed field format in Phase F7.

OP Code - Hexadecimal operation code for machine instructions (See Appendix B for internal assembler instruction codes).

ASC - Assembler switch codes. (Inserted in Phase F2 but not used by the macro generator phase. See the edited text record fixed field format in Phase F7.)

NL - Name length

OPL - Operation length

OPNL - Operand length

COML - Comments length

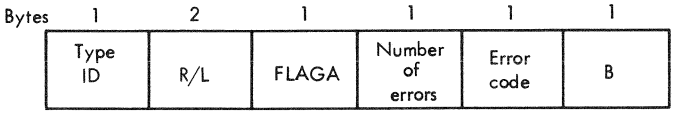
Text - Relevant text from source record, all source between beginning and end column, and from continue to end column, if continuation record. All blanks, except one, after end of comments field of last (or only) source record are dropped. As an exception, for macro prototype statements, one logical 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 ".*".

Error Record

This format is as follows:



Type ID - 08_{16} Warning message (same as source record)
$0D_{16}$ All other errors

R/L - Record length which must be between 0007_{16} and $002B_{16}$.

FLAGA - Always 10_{16}

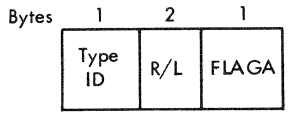
Number of Errors - Always 01_{16}, i.e., one record is created for each error.

Error Code - Type of error

B - Always 00_{16}

End of Data Set

This format is as follows:



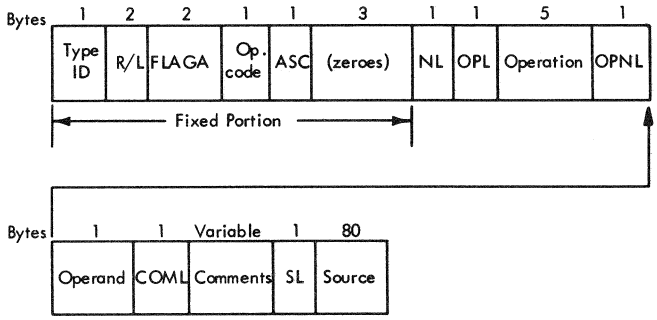
Type ID - $0A_{16}$

R/L - Record length, which must be 0004_{16}.

FLAGA - 00_{16}

Reproduction Record

This format is as follows:



The first nine bytes of the REPRO record are the same as the logical statement record (fixed portion).

NL - Length of name field. Always 00_{16}. (There is no name field in a REPRO instruction.)

OPL - Length of operation field. Always 05_{16}.

Operation Field - Always "REPRO".

OPNL - Length of operand field. Always 01_{16}.

Operand field - Always blank. (There is no operand in a REPRO instruction.)

COML - Length of comments field.

Comments field.

SL - Length of source record. Always 50_{16} ($=80_{10}$).

Source - The 80-byte source record to be reproduced.

SUBROUTINES

DRIVER

One card is read and control is transferred to DRVER1.

DRVER1

The name and operation field of each instruction is scanned. The name field is indicated as defined if non-blank. Syntax errors are noted and, at this point, may abort this statement.

If a prototype was expected, the name (if not already present) and SYSUT3 NOTE/POINT address is entered in the global dictionary, and each parameter (name field included) is scanned and entered into the local dictionary for this macro. Edited text is built up during scanning. The program goes to the output routine ENDOPR, and then to DRIVER. If a prototype was not expected, the operation field is searched for in the global dictionary. If not found, statement sequencing is checked and the operation field is entered in the global dictionary. The statement is treated as a macro instruction. The name field and all operands are edited and edited text is produced. The program goes to ENDOPR for each operand. When all operands have been processed, control goes to DRIVER.

If the operation code is found and is not an assembler operation, the statement sequencing is checked. The operand field is scanned and put into edited text format. Control goes to ENDOPR.

If the operation code is a declaration, the operand fields are scanned and errors diagnosed. Each operand is inserted into either the global dictionary or the local dictionary. Control goes to DRIVER.

If the operation code is a SET statement, it is checked for statement sequencing. If valid, the "a" pointer of the variable symbol is found in the dictionary and inserted into the edited text. Control then goes to ENDOPR.

If none of the above operation codes are found, the program performs a computed GO TO based upon the operation code.

AIF and AGO

The evaluation field, if present, is scanned and put into edited text. The "a" pointer of the sequence symbol is inserted into the edited text formats. Control then goes to ENDOPR.

ANOP, TITLE, MNOTE, MEXIT, EQU, CSD, CNOP, DROP, USING, ORG, PRINT, SPACE, PUNCH, ENTRY, COM, EJECT, AND LTORG.

All these subroutines follow a procedure similar to AIF and AGO. In each case, control goes to ENDOPR.

MACRO

Control returns to DRIVER.

ICTL

This subroutine puts out an error message and moves an END card image into the input buffer. Control is returned to DRVER1.

COPY

If in a system macro, the address of the library is NOTEd. In either case, the input in GETSRC is set to the library. A FIND to the library is then undertaken. Control returns to DRIVER.

ISEQ

The operand field is scanned and new input parameters in GETSRC are defined. Control goes to DRIVER.

REPRO

The next card is read and put into an edited text format. Control goes to ENDOPR.

EXTRN

The operands of EXTRN and WXTRN statements are scanned and indicated as defined for the dictionary. Control goes to ENDOPR.

START, DXD, DSECT, and CSECT

The type attribute is set, and control goes to ENDOPR.

DC, DS

The operand fields are scanned, and, if in open code, the attributes are defined. In either case control goes to ENDOPR.

CCW

Type, length, and scale attributes are moved into the edited text field. The operand field is scanned and control goes to ENDOPR.

MEND

This subroutine goes to ENDOPR to write edited text for the MEND card. It then goes to DCLSE.

END

The name, operation, operand, and comments are written in an edited text format. Control is transferred to DCLSE.

OPSYN

An error message is issued. Control is returned to DRIVER.

DCLSE

This subroutine forces out the local dictionary block on SYSUT3. It saves the NOTE/POINT address of the last local dictionary block and writes out the local dictionary block. It zeros the local hash table. It subsets the dictionary just written. The local dictionary on SYSUT2 is read in, subsetted, and written out on SYSUT3. Control goes to DRIVER if macros are being edited (i.e., a MEND card transferred control here); otherwise, it proceeds as follows.

If any undefined operation codes are present, (standard ICTL parameters are assumed), the input in GETSRC is set to SYSLIB, and a FIND of this undefined operation code is executed. If found, the program returns to DRIVER. If not found or if no undefined operation codes are present, the open code dictionary is subsetted, the global dictionary is subsetted, and a XCTL to Phase F3 is executed.

ENDOPR

The comments field is edited, and control is transferred to NDSMT3.

NDSMT3

If in a macro, this subroutine selects SYSUT3 as unit to write upon; otherwise, it writes on SYSUT1. If lookups are not suppressed, it inserts a little "a" pointer into the edited text format where appropriate. In either case, it writes edited text on the selected unit. If an error record was written on SYSUT3, it is also written on SYSUT1. If a return is expected because of a previously set switch (SWTCH7, bit 0), it returns; otherwise it goes to DRIVER.

GETSRC

This program reads a record from SYSIN or SYSLIB, depending upon which was selected. The source is immediately written on SYSUT1 if GETSRC is not processing system macros. Continuation cards and sequence numbers are noted and appropriate action is taken. Control returns to the caller.

BWRITE

This subroutine moves data to the buffer. If the buffer is full, it is written to the appropriate utility. In either case, control returns to the caller.

BWFORC

If room exists in the buffer, control returns to the caller. Otherwise, it goes to BWRITE.

BWNOTE

This program NOTEs the position of the selected utility and returns it and control to the caller.

PHASE F3 - CONDITIONAL ASSEMBLY AND MACRO GENERATION

OVERALL OPERATION (FLOWCHARTS 8-11)

Phase F3 reads text from SYSUT1. See Figure 8. This text includes source, error, and edited records. See Table 6. Source and error records are immediately written onto SYSUT2. MEND and MEXIT statements are not processed. The edited text for the main portion of the program and for all macro definitions, plus the subsetted global and local dictionaries, are used to generate one-for-one statements in edited text form and to perform conditional assembly. Assembler edited text records are produced and written on SYSUT2. When the end of text is reached, control is transferred to PHASE F7 via MAC and RTA.

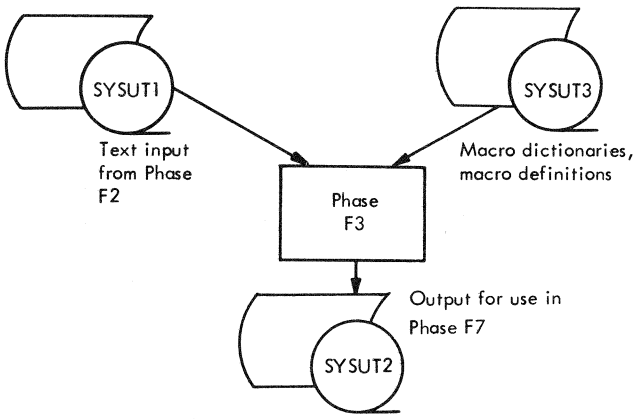


Figure 8. I/O Flow for Phase F3

PHASE F3E - ABORT CONDITION (FLOWCHART 12)

Phase F3E will be substituted for Phase F3 if any of the following abort conditions arises in Phase F2:

- The global dictionary fills up.
- The local dictionary exceeds 64K bytes on the overflow file or in core.
- The subsetting area is too small.

All text is read from SYSUT1. Only source records with accompanying error records are written on SYSUT2. All other records are bypassed. When the end of text is encountered, an edited text record for an END card is generated and written on SYSUT2, and control is passed to Phase F7 via MAC and RTA.

FUNCTIONS

The functions of Phase F3 are designed to:

- Initialize the phase.
- Evaluate conditional assembly expressions.
- Perform conditional assembly.
- Generate a parameter table from a macro instruction-macro prototype pair of statements.
- Generate assembler edited text records using macro definition edited text and the information in the global and associated local dictionaries.

Phase F3 evaluates conditional assembly expressions. If the expression was in a SETx statement, the "a" pointer associated with the SET variable symbol in the name field 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 repositioned and the appropriate text is read in and processed.

When a macro instruction is encountered, its exact location on the input file is NOTEd. The input file is SYSUT1 for outer macro instructions and SYSUT3 for inner macro instructions. A complete pass over the macro instruction text is made. Source and error records associated with the macro instruction are written on SYSUT2. When the end-of-macro instruction record is encountered, the position where the reading of text was discontinued is NOTEd so that the input of text can later be resumed at the correct position. The appropriate utility data set is then repositioned again to the beginning of the macro instruction text.

The "a" pointer associated with the macro instruction is used to inspect the associated 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, the first segment of the subsetted local dic-

tionary is read in. 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 F2, 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 is constructed at this location. This table indicates the values to be substituted for macro prototype symbolic parameters when they are referenced in model statements or inner macro instructions. &SYSNDX and &SYSECT are treated as parameters, and their entries are assigned the first two entries of this parameter table. The third entry is assigned to the name field. 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 cycle of comparisons to find the matching macro instruction operand corresponding to the next sequential prototype keyword begins where the last cycle left off. The operands are compared in a "wrap-around" 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. (This length table contains the accumulated length of each parameter entry. In effect, an entry in this table is an increment that must be added to the address of the beginning of the parameter table to locate its associated parameter entry.)

After the parameter table is completed, the rest of the macro definition fully-edited text is read in. 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, followed by the address of the length table and address of the beginning of 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 F2 to enter the local dictionary associated with the inner macro instruction. If there is not sufficient room left, this information is saved for diagnostic purposes, the concerned local dictionary is not brought into main storage, further (deeper) nesting of macro instructions is discontinued, and the input data set (SYSUT3) is NOTEd. Processing continues at the discontinued text of the next outer level macro.

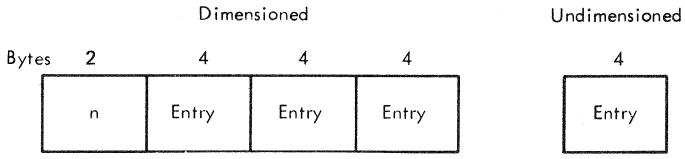
When the ACTR value is exceeded within a macro expansion, the information is saved for diagnostic purposes, control is returned to the outermost macro instruction expansion, and processing continues. If the ACTR value is exceeded in open code processing, the information is saved for diagnostic purposes, 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 (Phase F7) via MAC and RTA.

DICTIONARY ENTRIES

SETA Variable

This format is as follows:

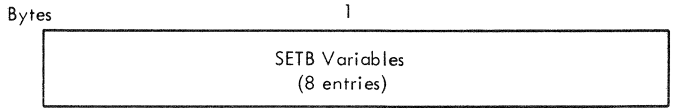


n - SET variable dimension (number of variables in entry). These bytes are not present if n=1 (an undimensioned SET variable).

Length of a complete entry is 4 bytes if the variable is undimensioned and 4n+2 bytes if it is dimensioned. There are 4 bytes per subentry.

SETB Variable (Non-Dimensioned)

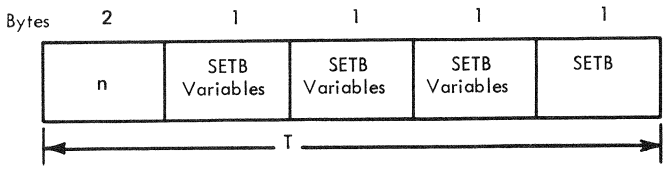
This format is as follows:



Each bit represents a SETB variable.

SETB Variable (Dimensioned)

This format is as follows:



$$T = 2 + \left\lceil \frac{n+7}{8} \right\rceil$$

where $\left\lceil \frac{n+7}{8} \right\rceil$ should be rounded to the next lowest integer.

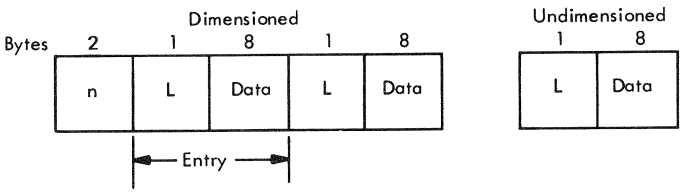
Example:

LCLB &B(26)

$$T = 2 + \left\lceil \frac{26+7}{8} \right\rceil = 2 + \left\lceil 4\tfrac{1}{8} \right\rceil = 2 + 4 = 6$$

SETC Variable

This format is as follows:



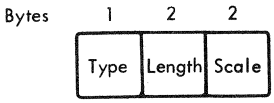
Length of entry is 9 bytes if the variable is undimensioned and 9n + 2 bytes if it is dimensioned.

n — Same as SETA variable.

L — Length of character string (data), (true length).

Table Format for Symbols

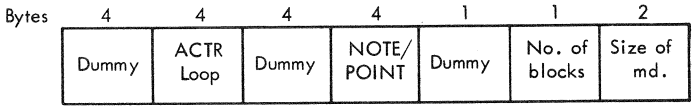
A symbol entry will always contain an L' and S' entry. If there is none, these entries will contain the value of zero. The format is as follows:



Type — Type Attribute

Macro Dictionary Header

This format is as follows:



The header is attached to the subsetted dictionaries' output by Phase F2.

Dummy — Not used.

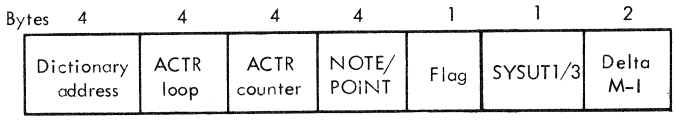
ACTR Loop — The A counter loop limit, i.e., initial assumption made by processor.

NOTE/POINT — Location of block on SYSUT3 in which the macro definition fully-edited text begins (points to prototype record).

No. of blocks — The number of segments on SYSUT 3 in which the dictionary is contained.

Size of MD — The size of the total macro dictionary in bytes.

The header as modified during F3 processing of the macro instruction is as follows:



DICT ADDR — The location of the 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 — The current loop count, i.e., the number of times the loop has been passed.

NOTE/POINT — The location of the block in which the end-of-macro instruction record is located.

FLAG — A switch, used to signal whether the length table has already been stored following a parameter table.

SYSUT1/3 -- A switch to indicate:

8 — Input is from SYSUT3

0 — Input is from SYSUT1

Delta M-I -- Position of discontinued text (record following macro instruction) relative to beginning of block.

Macro Dictionary Parameter Entries

The macro dictionary parameter entries are given in Table 7. The SET variables are the same as given above.

Table 7. Parameter Entries

0	No. of M-1 param. (0)	&SYSNDX	L	4-Byte Char. Value			4-Byte Binary								
1	No. of M-1 param. (1)	&SYSECT	L	BCD Name											
2	T	CHAR FLAG	L K^1	Char. String											
3	T	HBD FLAG	3-Byte Binary			Char.	L K^1	BCD 'VALUE'							
4	T	CSD FLAG	3-Byte Binary			Char.	L K^1	BCD 'VALUE'							
5	T	SYM FLAG	5-Byte Attributes			Char.	L K^1	BCD NAME							
6	T	SUB FLAG	2 Total L of Entry	2 K^1	1 N^1	(	2 $T.L_1$	Entry 2-5	,	$T.L_2$	Entry 2-5	)	N		

L_1

T = Type Attribute (see Table 9).

TL_1 = L

K^1 = No. of characters in an operand (excluding commas)

= No. of characters between outer commas in a sublist

$= \sum_{2}^{n} TL + N^1 + 1$

N^1 = No. of operands of a sublist

= 1 (Does not equal sublist)

= 0 (Omitted operand)

CHAR = Character string

HBD = Hex, binary, decimal self-defining term

CSD = Character self-defining term

SYM = Ordinary Symbol

SUB = Sublist

&SYSNDX = Macro-instruction Index

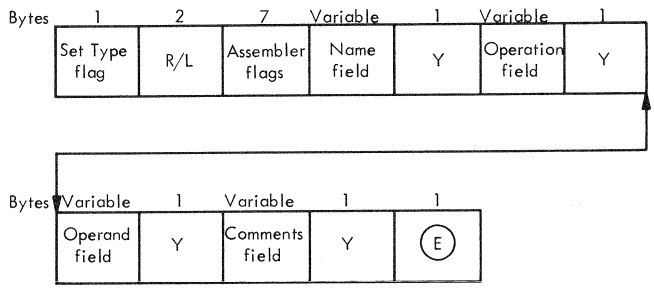
&SYSECT = Current Control Section

INPUT RECORD FORMATS

All flags used in the input record formats are listed in Tables 6, 8, and 9.

Machine Instructions

The general format of machine instructions is as follows:



- The text consists of normal "PUT" action (no evaluation) and/or evaluation text (see VALUAT subroutine).
- Each field is terminated by an end-of-field flag (Y).
- All fields except the comments field may contain evaluation text.
- Output to the assembler will contain four fields, a length followed by a character string for each field.

Table 8. Assignment of Flag Values (Phase F3)

Value (Hex.)	Flag Description
00	Period
01	Right Paren.
02	Left Paren.
03	Subscripted Left Paren.
04	Plus
05	Minus
06	Multiply (asterisk)
07	Divide (slash)
08	Equal
09	Not Equal
0A	Less Than
0B	Greater Than
0C	Less Than or Equal to
0D	Greater Than or Equal to
0E	Not
0F	Or
10	And
22	Hexadecimal Self-Defining Term
23	Binary Self-Defining Term
24	Decimal Self-Defining Term
25	Character Self-Defining Term
26	Null Symbol & Evaluation Flag
27	Character String
28	SETA
29	SETB
2A	SETC
2B	Comma
2C	Begin Substring
2D	Begin Substring Operands
2E	First Operand Completed
2F	Second Operand Completed
30	Actual Internal Value Right Paren. Used Only on Sublist

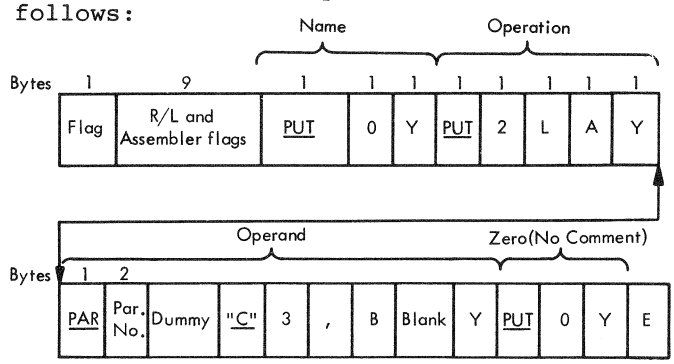
Table 8. Assignment of Flag Values (Phase F3) (Cont'd)

Value (Hex.)	Flag Description
31	Arithmetic Expression mode.(Absence indicates character expression)
32	Blank
33	Type Attribute Reference
34	Length " "
35	Integer " "
36	Scale " "
37	Number " "
38	Count " "
39	Symbolic Parameter Reference
3A	System List
F0	Sublist
F8	End of machine instruction field (Y)
F9	Continue sublist
FA	Symbol
FB	Positional (P)
FC	Keyword (K)
FD	PUT (No evaluation necessary)
FE	End of block
FF	End of evaluation (E)

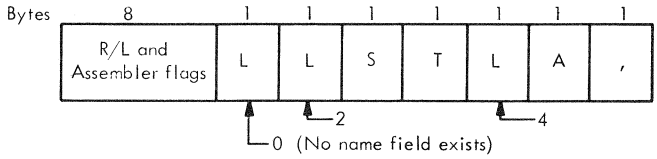
●Table 9. Phase F3 Internal Values for Type Attributes

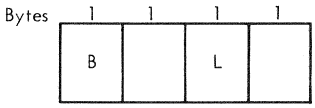
Value (Hex.)	Type	Value (Hex.)	Type
00	P	0D	W
01	Z	0E	I
02	E	0F	C
03	D	10	Q
04	K	11	B
05	F	12	J
06	G	13	X
07	H	14	M
08	S	15	T
09	A	16	U
0A	V	17	O
0B	Y	18	N
0C	R	19	U'
		1A	L
		1B	$

An input record example (Load Address) follows:



A typical output record (Store) is as follows:





Y — End-of-field flag (X'F8')

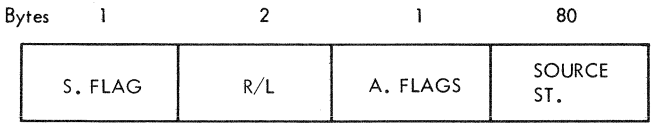
E — End-of-statement flag (X'FF')

NOTE: If an error is detected during evaluation, a full word of zeros is put out to assure the assembler four fields. An error record will then be put out by the generator phase.

NOTE: CSECT statements have the same format as the statement type flag, except for the first byte.

Source Statements

The source format is as follows:



S. FLAG - Source flag (X'08')

R/L - Record length (always $84_{10} = 0054_{16}$)

A.FLAGS - 1-byte assembler flags

Bit 0 Unused

Bits 1-3 Record type
000 — Source without editing = print as is
001 — Error
010 — Construct for print
011 — Construct for print if generated
100 — Process only
101 — Illegal
110 — Process and construct for print
111 — Process and construct for print if generated

Bit 4 Unused

Bit 5 Error record indicator
0 — No error record follows
1 — Error record follows

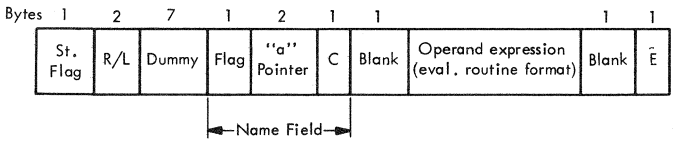
Bit 6 Continuation bit
0 — Source not continued
1 — Source continued

Bit 7 Unused

SOURCE ST. — Source statement

Set Statement

The set statement format is as follows:



ST. FLAG — SET statement flag (X'04')

Flag — SET variable flags (SETA, SETB, SETC)

C — Bit 0 Dictionary bit

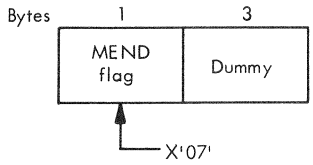
0 — Local
1 — Global

Bits 5-7 SETB bits

E - End of statement flag

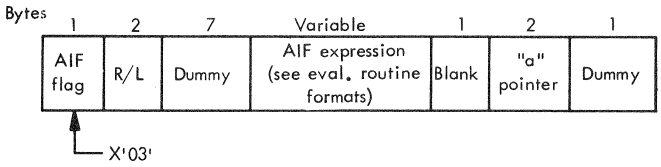
MEND or MEXIT

This format is as follows:



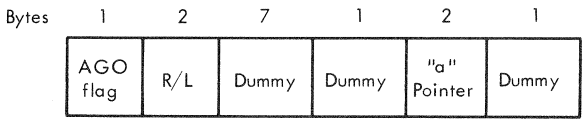
AIF Statement

This format is as follows:



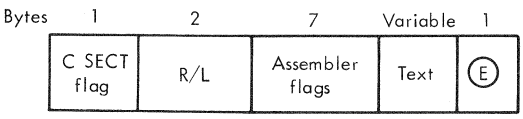
AGO Statement

The AGO format is as follows:



CSECT, DSECT, START

This format is the same as that for Machine Instruction, above.

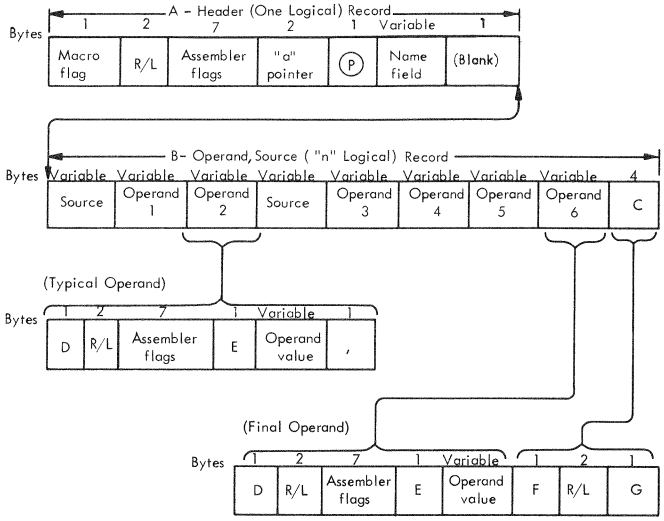


Error Statements

Error statements are treated exactly the same as the source statements with the same format. However, the error statement flag is different from a source statement, because error statements must be distinguished from source statements in processing macro instructions and prototypes. The record length is seven bytes.

Macro Instruction

General Format. The general format is as follows:



A — One logical record. It contains the macro flag, the "a" pointer, and repeats the name field as the first operand. The format of the name field is that of an operand.

B — "n" logical records. These logical operand records can contain source records between any operand records. (There are six shown in this example.)

C — End of macro instruction record.

D — Operand record flag (X'0B').

E — Positional or keyword flag. A positional flag must precede each positional parameter. A keyword flag must precede each keyword parameter. Positional parameters must precede keyword parameters.

(P) = Positional (X'FB')

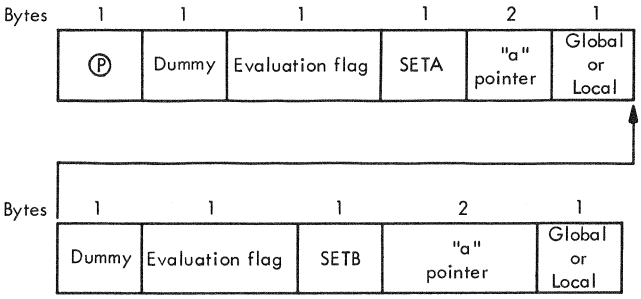
(K) = Keyword (X'FC')

F — End of macro instruction flag (X'0C')

R/L (End of machine instruction record) — Record length = 4

G — No. of keyword operands in the macro instruction.

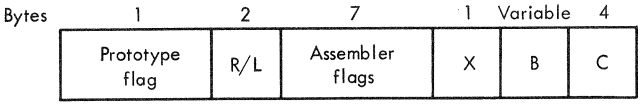
Continuation and Segmenting. Macro instructions (and prototype) may be segmented by adding a continuation flag after the last operand record in the block. An operand record must be fully contained in one block.



NOTE: Only one evaluate flag is needed at the beginning of the parameter. However, it will be ignored if repeated.

Prototype Statement

The prototype statement format is as follows:



X — Number of positional operands.

B — Same source and operand mix possibilities as macro instruction, except that only keyword operands appear in prototype.

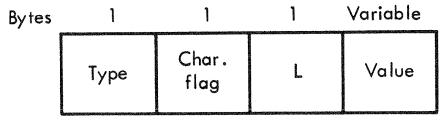
C — Same as end of macro instruction record.

NOTE: Trailer record "C" would contain number of keywords for prototype.

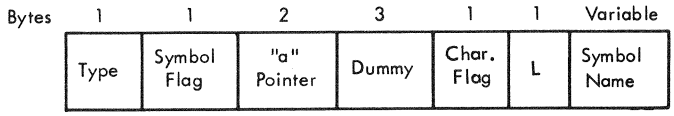
Operand Value Formats

Each operand value must be preceded by a Ⓟ or Ⓚ type. There is one exception. See Sublists. Formats are as follows:

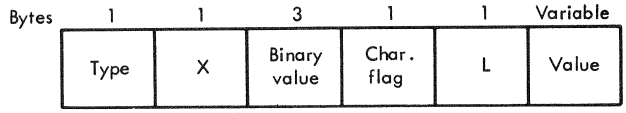
1. Character



2. Symbol

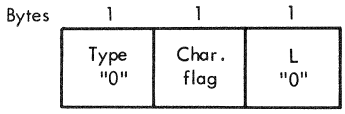


3. Hex, binary, decimal self-defining terms

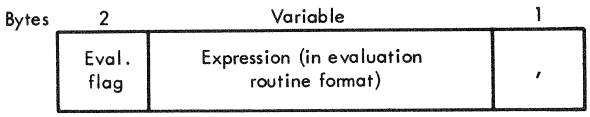


X — Flag (Hex, binary, decimal)

4. Character self-defining terms.
   Same as 3, except that X = CSD flag.
5. Omitted operand
   This is a character operand whose L = 0.



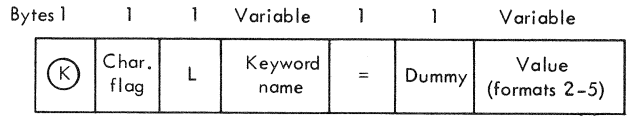
6. When the evaluation flag is in the record, operands are evaluated by substitution and concatentation. The evaluation flag precedes parameters which require evaluation, such as those containing variable symbols.



Parameters which do not require this flag are as follows.

Type	Parameters
HBD	Hex, Binary, Decimal
CSD	Character Self-Defining
SUB	SUBlist
'C'	Character
SYM	SYMbol

7. Keyword

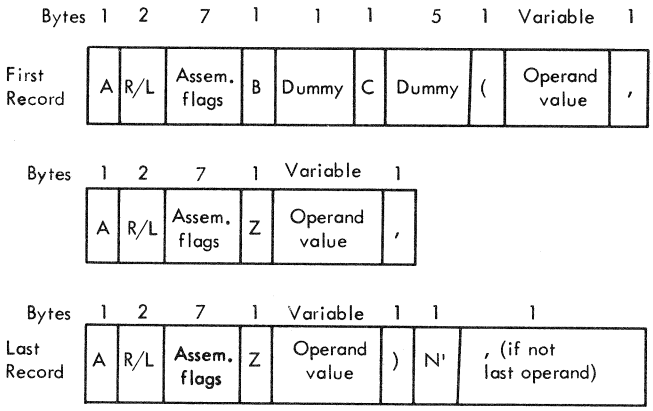


L — Length of keyword name + 1

The value of a keyword may also be a sublist. For prototype operands, only formats 1 and 4 above apply.

Sublist Operands

Sublist operand formats are as follows:



Source records may appear between sublist operands.

Operand value — use formats 2-5 above.

B — The positional (P) or keyword flag (K) which appears only this time for the entire sublist operand.

C — Sublist flag (X'F0')

A — Operand record flag (X'0B')

Z — Continue sublist flag (X'F9')

N' Number of operands in sublist.

EVALUATION ROUTINE FORMATS

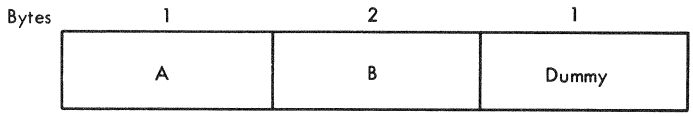
Operand Reference

Reference to any operand in all input statements is made by position. Operands are numbered as follows:

0 - &SYSNDX
1 - &SYSECT
2 - Name field
3 - Operand field

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.

Example:

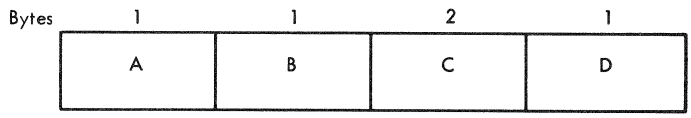


A — Operand request flag (X'39')

B — Operand number

Attributes (L',I',S',T')

Attribute formats are as follows:



A — Flag byte (type of attribute)

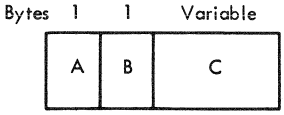
B — Parameter or symbol flag

C — 2-byte pointer for symbol. The parameter position in the 2nd byte (B) is for the parameter.

D — Dummy

Character String (C' ') or ' '

This format is as follows:



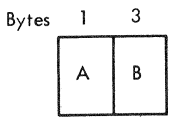
A — Flag byte

B — True length byte

C — Data bytes (n bytes of characters). This does not exist if B = 0.

Decimal, Hex, Binary Value, Character Self-Defining Terms

This format is as follows:

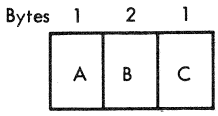


A — Flag byte

B — Data bytes (3 bytes of data in binary)

Variable Symbol

This format is as follows:



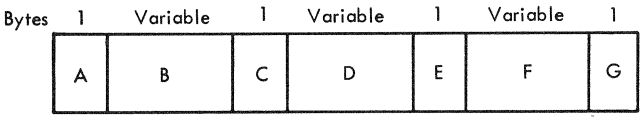
A — Flag byte (SETA, SETB, SETC)

B — 2-byte pointer

C — Bits 5-7 = Bit no. for SETB
0 = 0 for local, 1 for global.

Substring

This format is as follows:



A — Substring flag

B — Character expression

C — Sublist left parenthesis flag

D — Expression 1

E — Sublist comma flag

F — Expression 2

G — Sublist right parenthesis flag

Subscripting

In this format the left parenthesis is replaced by a special subscript left parenthesis flag (X'03). The rest of the format is as previously described.

Concatenation

Concatenation occurs automatically by eliminating the period, if it exists. Two character strings (or set variables), one immediately following the other, will be concatenated, and no concatenation flag is required.

FUNCTIONAL PROGRAM SECTIONS AND ROUTINES

IEUF3

This routine relocates permanent and open-code dictionaries, initializes I/O buffers, and initializes input pointer and macro base pointer.

CGOTO

This is a computed GO TO on the various statement types.

MACHOP

This routine processes edited text records, evaluating fields which require substitution.

SOURCE

This routine puts out a source or error record.

SETST

This routine determines the address of the result field, evaluates the operand, and stores the result.

CSECT

This routine stores the last CSECT name and outputs the record.

AIFST

This routine evaluates the expression. If expression is false, processing continues with the next sequential record. If the expression is true, the text file is repositioned to the sequence symbol.

AGOST

In this routine the text file is repositioned to the sequence symbol.

MENDST

If exit is from an outer macro, this routine sets the text file to read from SYSUTI; otherwise it continues reading from SYSUT3. It positions the text file to read from discontinued text.

MINSTR

This routine outputs the source, notes the discontinued text, and repositions the text file to the beginning of the macro instruction.

ENDST

This routine damps output buffers and rewinds utility files.

BEGMAC

This routine reads the macro dictionary and initializes the parameter table with SYSNDX and SYSECT entries.

PROTO

This routine points to macro definition prototype and reads it into the input buffer.

PROTO1

This routine builds the parameter table, evaluating operands as needed. A merge of keywords is performed if keywords exist.

VALUAT

This routine initializes the mode switch to character expression, pointers, operator table, and result list with zero and zero-out length buckets of string areas. In general, this routine is called for the evaluation of an expression. Evaluation may be required in the name, operation, or operands fields. An expression may be a simple parameter reference or a complicated arithmetic expression in a SETA operand.

Any term that requires a retrieval from some dictionary (parameter reference or SET variable) will eventually pass through the VALUAT routine.

The routine is designed to operate on the flags and operators shown in Table 8.

SYMBL

This is an input pointer pointing to an operator.

CHFORC

This routine tests for the type of operator.

ADVOP

There is no forcing in this routine. It advances the operation pointer and the emergency operation into the operation table. If the operation is OR or AND, it reinitializes the mode switch to character expression.

ADVINP

This routine advances the input pointer.

FORCE

This routine determines if the new operation forces the last operation entered in the operation table.

TSTOP1

This routine tests for end of expression. If so, it returns to the driver routine.

DOOPR

This routine fetches the address of the two fields to be processed.

SUBSC

This routine processes subscripted variable or parameter sublist or SYSLIST.

RELINT

This routine initializes a string pointer with the address of string area 2 and turns off the 'PUTST' switch.

META3

If character expression mode exists, this routine fetches the binary word and converts it to decimal. If arithmetic expression mode exists, it initializes for entry of the result, then stores the result.

MEB4

If in the arithmetic expression mode, this routine initializes for entry and stores the result.

METC4

This routine fetches the length of a string, initializes for the entry of the result, and stores the result.

DOOPR

This routine processes the relational operator and stores the result.

RELAT

This routine processes the relational operator and stores the result.

ARITOP

This routine processes the arithmetic operator and stores the result.

NOTOPR

If outside range of valid flags, this routine sets the end of expression flag in the operation table; otherwise it performs a computed GO TO on the flag.

CSD

This routine translates a character string back to the original representation.

DECINT

This routine stores the value of a decimal in the intermediate result list.

META

This routine initializes for a SET variable.

METB

This routine initializes for a SET variable.

METC

This routine initializes for a SET variable.

CHARST

This routine puts a string in the string area.

BEGSUB

This routine stores the length of a string already present in the string area and sets the substring mode.

SETARE

This routine sets the Mode switch to the arithmetic expression mode.

SBEND

This routine sets the substring comma or the left parenthesis switch.

TATTBT

This routine checks for the type attribute of a parameter.

LATTBT

This routine checks for the length attribute of a parameter.

SATTBT

This routine checks for the scale attribute of a parameter.

PACK3

This routine stores the address of a result in the pointer list.

SYSLST

This routine stores a parameter flag to simulate a parameter reference.

PARMTR

This routine stores a parameter flag and number into the pointer list.

ATTPAR

This routine checks for the attribute of a parameter.

OVERALL OPERATION (FLOWCHARTS 13-27)

Phase F7 has three general functions:

1. Processing symbols
2. Processing literal values
3. Assigning storage locations

Symbols are processed by entering the mnemonics and their relative storage addresses in a symbol table. Addresses are assigned relative to the beginning of the control section in which they are determined. (While the program performs this function, it is working in the "assignment mode.")

If more symbols are defined in the user's program than will fit in the main storage spaces allotted for the symbol table, the point in the text that caused overflow is NOTEd and the remainder of the program is processed without making further symbol table entries. However, symbols already defined in the symbol table are substituted into the text where applicable. (While performing these functions, the program is in the "substitution mode."

When the end of the text data set is reached, it is TCLOSEd to the beginning, and operand processing continues until the previously noted overflow position is reached. At this point, the mode of processing reverts to making symbol table entries (assignment mode), and the next symbol table segment is constructed.

The process of building symbol tables and processing operands continues until the last symbol defined has been placed in the symbol table.

As name fields are being processed, Phase F7 collects appropriate symbols and creates an external symbol dictionary (ESD) which will be processed by Phase F Interlude (FI).

Phase F7 also processes literals and self-defining terms in expressions affecting the location counter. Literals are entered in the symbol table. When an LTORG or END assembler instruction is encountered, the literals in the table are inserted in the program stream.

All mnemonic operation codes created by concatenation or parameter substitution during macro generation are translated in this phase. (If the DOS assembler option is specified, F7 treats CXD, DXD, and OPSYN assembler instructions and Extended Precision machine instructions as undefined. It treats L-type and Q-type DC and DS instructions as unknown types.)

Cross-reference records are generated in Phase F7 during symbol processing and expression evaluation.

Phase F7 also creates a TESTRAN symbol table if requested by the programmer, and writes cards generated by PUNCH and REPRO statements.

Intermediate text is read from SYSUT2, and literal pools and intermediate text are written on SYSUT1.

Overflow of any of the following tables will result in writing overflow segments on SYSUT3.

- External symbol dictionary table segment.
- Literal-pool base table segment.
- Cross-reference table segment.

I/O FUNCTIONS

Phase F7 may make several passes at the source text. See Figure 9. On the initial iteration, the text is read from SYSUT2 and reblocked onto SYSUT1. On subsequent iterations, the text is passed between SYSUT1 and SYSUT2.

The text is reformatted such that a "broken" record will always start at the beginning of a physical block. This will ensure that a logical record will be contained within a single physical block.

Edited text records are moved from the input buffers to a work area where work buckets are attached. The records are then transferred to the output buffers for eventual output to a utility file.

Error records are generated in text format and are transferred to the output buffers as errors are encountered.

As the cross-reference block and literal base table overflow, they are written onto the overflow file, SYSUT3. Each time the

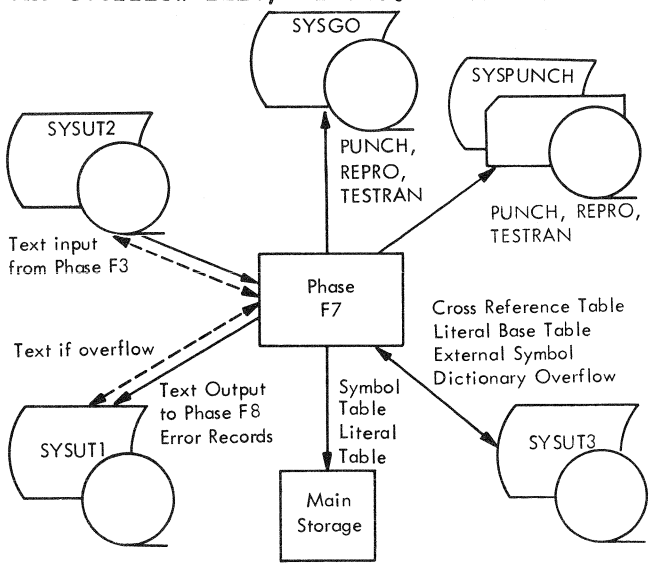


Figure 9. I/O Flow for Phase F7

external symbol dictionary overflows it is written onto SYSUT3, and its position on the file is NOTEd for future reference within the phase. The external symbol dictionary blocks are identified and called from the overflow file through the use of the NOTEd record position.

RECORD FORMATS

Cross Reference Records

This format is as follows:

Bytes	8	1	2	2	3	1
	Symbol	FLAGA	Statement No.	Length Attribute	Value	FLAGB

FLAGA -
$F0_{16}$ - Base symbol (type 1)
$F1_{16}$ - Reference to symbol (type 2)
$F2_{16}$ - Multiply defined symbol (type 3)

FLAGB -
0 - Absolute value
Not 0 - External symbol dictionary ID

Error Record

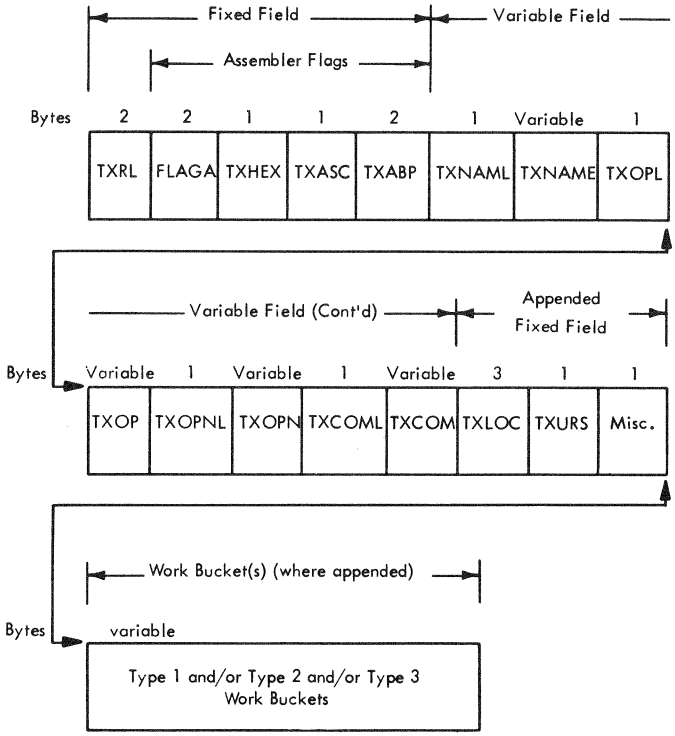
This format is as follows:

Bytes	2	1	1	1	1
	R/L	Flag	Error count = N	Error type	Column pointer

R/L - Record length. R/L = 4 + 2N, where 16 ≥ N>0. (There may be as many as 16.)

Edited Text Records - General

Phase F7 receives input statements in a format prepared by the macro generator. The formatted text records are called "Edited Text Records." Phase F7 processes these records and attaches an "appended fixed field" and, when required, "work buckets". The format of the edited text record is as follows:



The fixed field and the variable field are the input record to Phase F7. The appended fixed field is attached to this record in Phase F7.

Edited Text Record Fixed Field Format

TXRL - Record Length. This will be set to the total number of bytes (including appended fields) by the GET statement routine in Phase F7.

FLAGA - Flag byte. Contains the following bits:

Bit 0 TXRLI. Last record in buffer indicator. Correctly set by the GET statement routine in Phase F7.

Bits 1-3 TXRT. Record type
000 - Print as is. Source record only. (These are assembly records created from program source input records in Phase F2.)

001 - Error record. (Created in any phase.)
010 - Print as is, but do not display a statement number. Source record only. (Created in Phase F3 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 F2, or may be source record created in Phase F3 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 F2.)
101 - Internal assembler control record. (In Phase F7, CSECT, ORG, and LTORG.) Edited text records are generated for END statement processing.
110 - Process this record and construct source record for print. (Edited text, but no source, e.g., Phase F7 literals, and Phase F3 conditional assembly substituted statements outside of macro definitions.

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 F3.)

Bit 4 TXBF. Break flag. Indicates that a logical record continues in the next physical block. The Phase F7 GET routine arranges all edited source and edited generated records so that this condition does not exist.

Bit 5 TXERI. Error record follows indicator. Used by the PUT ERROR common subroutine in phases F7 and F8 to determine whether to create a new error record or to attach to an existing error record.

Bit 6 TXESI. Equal sign indicator. Set by the Phase F7 GET statement routine.
0 - There is no literal in the operand of this statement.
1 - There is a literal in the operand of this statement.

Bit 7 TXMARK. Phase F7 iteration point flag.

Bits 0-1 TXTO. Type of operation.
01 - Machine Operation.
10 - Assembler operation.
00 - Unchecked. (Phase F7 GET statement will set equal to 01 or 10 if a 00 condition exists and a legal operation code is converted.)

Bit 2 TXEMF. Extended mnemonic flag.

Bit 3 TXMDN. Multiply defined name indicator. (Set by Phase F7 for future passes.)

Bits 4-7 TXRIM. R1 mask for extended mnemonics. Used for special switch codes on assembler operations.
Bit 4 - Name required
Bit 5 - Name not allowed
Bit 6 - Operand required
Bit 7 - Operand not allowed

TXHEX - Machine operation code or internal assembler operation code.

TXASC - Assembler Switch Code for machine operations
Bits 0-1 00 - No special register requirements
01 - Register must be even
10 - Register must be 0 or 4
11 - Register must be 0, 2, 4, or 6
Bits 2-3 00 - No boundary alignment
01 - Half word
10 - Full word
11 - Double word boundary alignment
Bits 4-5 Operand format within instruction class
Bit 6 1 - Literal permitted in 2nd and 3rd operand.
Bit 7 1 - Literal permitted in 1st operand.

TXASC - Assembler Switch Code for Assembler Operation
Bit 0 Uninitiated private code.
Bit 1 Possible symbol table entry.
Bit 2 Location counter reference.
Bit 3 Special Phase F7 cross-reference.
Bit 4 Substitution required.
Bit 5 Not Used
Bit 6 Not Used
Bit 7 Phase F8 uninitiated private code.

TXABP - Appended fixed field pointer.

Edited Text Record Variable Field Format

TXNAML - Name field length. If zero, there is no name field.

TXNAME - Name field.

TXOPL - Operation field length.

TXOP - Operation field.

TXOPNL - Operand field length. If zero, there is no operand field.

TXOPN - Operand field.

TXCOML - Comments field length. If zero, there is no comments field.

TXCOM - Comments field. This field will contain comments and extraneous data.

Edited Text Record Appended Fixed Field Format

TXLOC - Location counter. Set by Phase F7 during assignment pass.

TXURS - Unresolved symbol counter.

Misc. -

Bits 0-3	Unused.
Bit 4	TXLES. End of string indicator for literal DCs. Unused on all other types.
Bits 5-7	TXSTG. String number for literal DCs.
(or)	TXALIN. Alignment for machine operation codes.

Work Buckets

There are three primary types of work buckets:

- Type 1 - Literal in operand.
- Type 2 - Symbol in operand.
- Type 3 - DC, literal DC, and DS operation code.

These work buckets are appended to edited text records by the Phase F7 GET statement the first time through.

Literal in Operand. If the equal sign indicator is set, a 6-byte Type 1 work bucket will be appended immediately following the appended fixed field. The format for the Type 1 work bucket is given in Figure 10.

Byte 1 -

Bit 0	TXWTYP. Work bucket type (must be zero; 0 = type 1).
Bit 1	TXWLEN. Work bucket length (must be zero; 0 = 6 bytes).
Bits 2-5	(Blank)
Bit 6	TXLEVI. Literal evaluated indicated.
Bit 7	TXLASI. Literal assigned indicator.

TXLSTG - Literal string number. Corresponds to entry in literal base table.

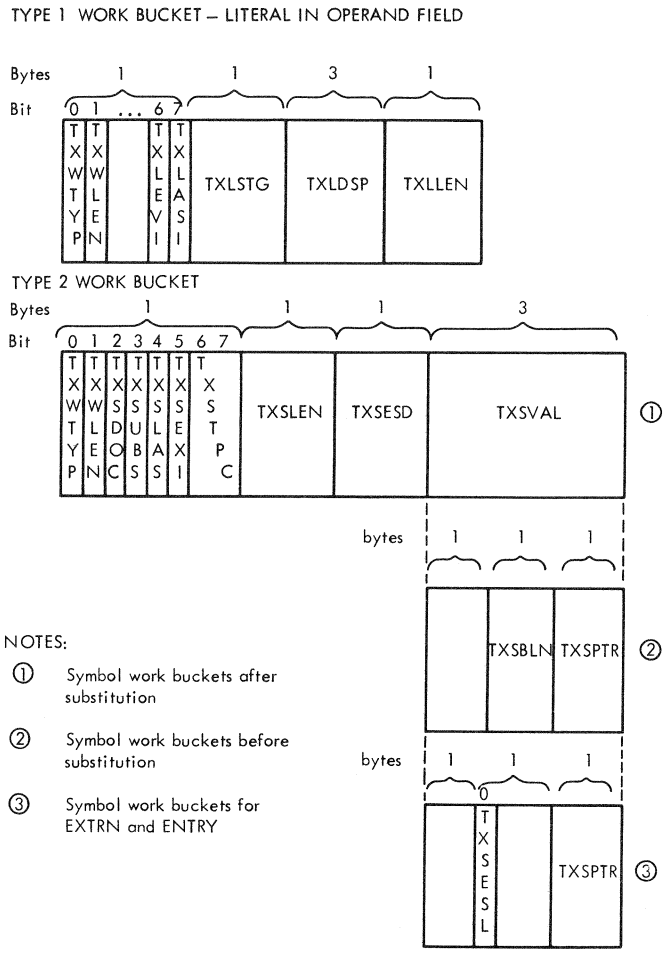


Figure 10. Types 1 and 2 Work Buckets

TXLDSP - Literal string displacement. If the value substituted indicator equals zero, the third byte of TXLDSP will contain a pointer to the symbol (relative to the beginning of the operand field). The second byte will contain the symbol length.

TXLLEN - Literal length attribute.

Symbol in Operand. If symbol table overflow occurs, it is necessary to append one Type 2 work bucket for each symbol in the operand field, including symbols within literal specification fields. See Figure 10. The order of work buckets corresponds to the order of the symbols in the operand field. CSECT, DSECT, and COM records will also be appended by a six-byte symbol work bucket.

Byte 1 -

Bit 0	TXWTYP. Work bucket type (must be one; 1 = type 2).
Bit 1	TXWLEN. Work bucket length (must be zero; 0 = 6 bytes).

Bit 2 TXSDOC. Symbol defined in DSECT or COM indicator.
Bit 3 TXSUBS. Value substituted indicator.
Bit 4 TXLAS. Last symbol in operand indicator.
Bit 5 TXSEXI. "Implied length exceeds 256" indicator.
Bits 6-7 TXSTPC. Adjective code.

TXSLEN - Implied length.

TXSESD - External symbol dictionary ID.

TXSVAL - Value. If the value substituted indicator equals zero, the third byte of TXSVAL will contain a pointer to the symbol (relative to the beginning of the operand field). The second byte will contain the symbol length.

TXSBLN - Symbol byte length.

TXSPTR - Pointer to symbol in operand field.

TXSESL - Last operand in EXTRN/ENTRY indicator.

DC, Literal DC, and DS Operation Code. If the operation code is one of these three types, one 15-byte Type 3 work bucket will be created for each operand. See Figure 11. Each operand work bucket will be followed by a six-byte work bucket for each symbol in the operand.

Byte 1 -
Bit 0 TXWTYP. Work bucket type (must be zero; 0 = type 3).
Bit 1 TXWLEN. Work bucket length (must be one; 1 = 15 bytes).

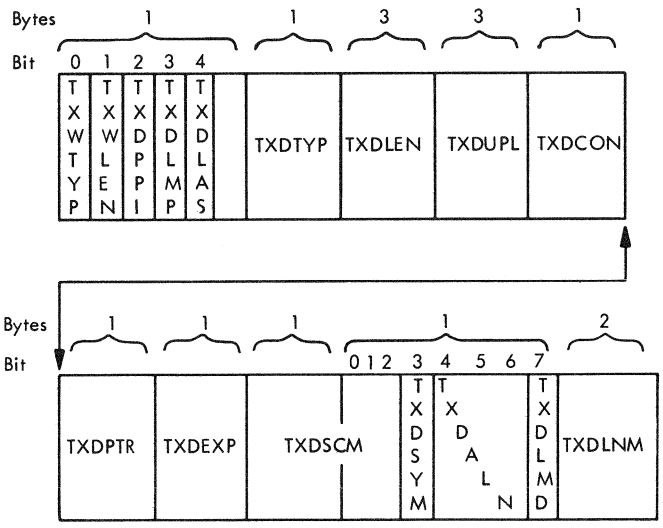


Figure 11. Type 3 Work Bucket

Bit 2 TXDPPI. DC previously processed indicator.
Bit 3 TXDLMP. Length modifier present indicator.
Bit 4 TXDLAS. Last operand indicator.
Bits 5-7 (Blank).

TXDTYP - Type, translated. See Table 10.

TXDLEN - Total length.

TXDUPL - Duplication factor.

TXDCON - Number of constants.

TXDPTR - Pointer to first byte of operand in text (relative to beginning of operand field).

TXDEXP - Exponent.

TXDSCM - Scale modifier.

TXDSYM - Symbol work buckets flag.

TXDALN - Alignment.

TXDLMD - Length modifier type.

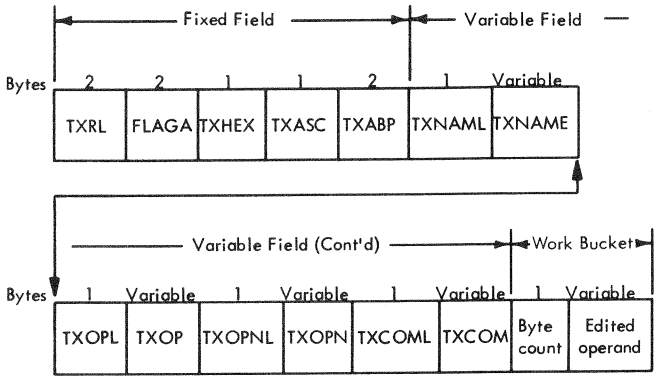
Table 10. DC/DS Type Indicators for Type 3 Work Buckets

Hexadecimal Number	Meaning
00	Character
01	Hexadecimal
02	Binary
03	Packed
04	Zoned
05	Double precision floating point
06	Single precision floating point
07	Full-word fixed point
08	Half-word fixed point
09	A-CON (Address Constants)
0A	Y-CON (Address Constants)
0B	V-CON (Address Constants)
0C	S-CON (Address Constants)
0D	Q-CON (Address Constants)
0E	Extended precision floating point

0 - Byte
1 - Bit

TXDLNM - Length modifier value.

Special Work Bucket. A special work bucket is used for TITLE, PUNCH, REPRO, and MNOTE edited text records. This format is as follows:

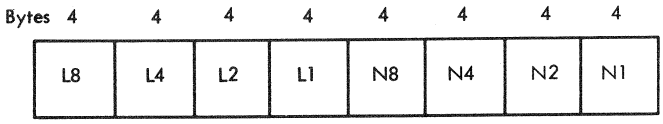


The eight-byte fixed field is the same as that described under "Edited Text Record Fixed Field Format." The variable field is the same as that described under "Edited Text Record Variable Field Format." However, in place of an appended fixed field is a special work bucket, as follows:

Byte Count - The byte count of the edited operand for punching or printing.

Edited Operand - The punch or print image in external code constructed from normal edited text in Phase F7. However, if PUNCH or REPRO is output in Phase F7, the byte count of this field is zero.

LTORG Statement Workbucket. The format for this workbucket is as follows:



L8 - Total length of 8-byte chain.

L4 - Total length of 4-byte chain.

L2 - Total length of 2-byte chain.

L1 - Total length of 1-byte chain.

N8 - Number of entries in 8-byte chain.

N4 - Number of entries in 4-byte chain.
N
N2 - Number of entries in 2-byte chain.

N1 - Number of entries in 1-byte chain.

TABLES

Symbol Table

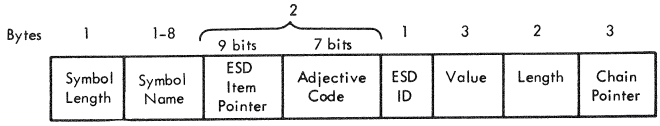
The symbol table contains a collection of symbols, OPSYN definitions, and literals with their associated attributes. It is built during Phase F7.

The symbol table remains in core storage as long as the space allocated will hold it. It is used by Phase F7, Phase FI, and Phase F8.

There are three types of entries in the symbol table.

1. Name entries.
2. Literal entries.
3. OPSYN definitions.

Name Entries. EQU, CCW, DC, DS, machine instructions, and LTORG, and external name entries EXTRN, WXTRN, START, CSECT, and DSECT.



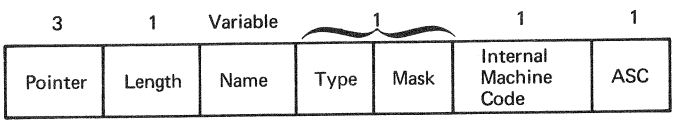
Adjective code -
Bit 1 1 - EQU operand defined as external symbol
Bit 2 1 - Pointer present.
Bit 3 1 - XD complete (external definition)
Bit 4 1 - LD complete (label definition)
Bit 5 1 - Defined in DSECT or COM.
Bits 6-7 External symbol dictionary type
00 - CSECT
01 - EXTRN/WXTRN
10 - DSECT
11 - NAME

Value - present only in name entries.

Length - present only in name entries.

Chain pointer - present only when a symbol with the same hash has been previously entered in the table. This pointer is the address of the previous entry.

OPSYN Entries. The format is as follows:



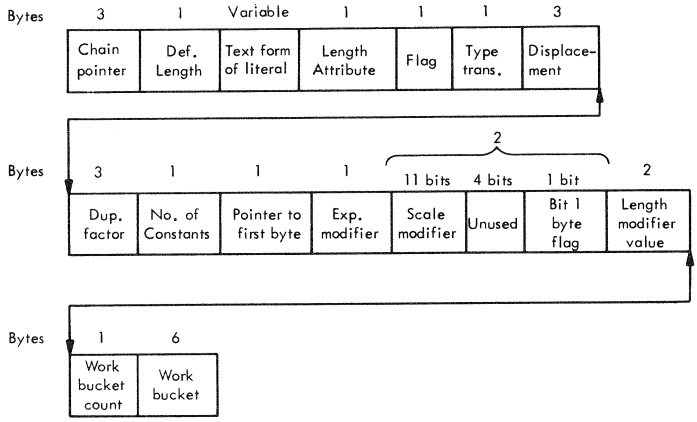
Chain pointer - present only when an OPSYN entry with the same hash has been previously entered in the table. This pointer is the address of the previous entry.

Type - attribute. See Table 6.

R1 Mask - R1 field for extended mnemonics. The extension of op codes field is omitted for machine operation code entries.

ASC - Assembler Switch Code. (See Edited Text Record Fixed Field Format.)

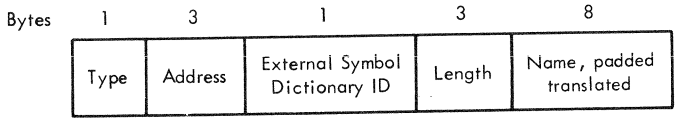
Literal Entries. This format is as follows:



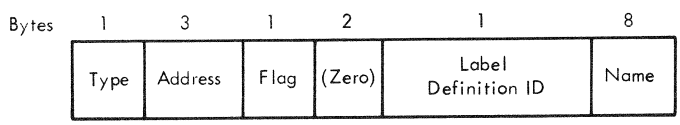
External Symbol Dictionary

External symbol dictionary items are generated by START, CSECT, private code, COM, DSECT, external dummy sections, ENTRY, EXTRN, WXTRN, and V-type DC instructions. Formats are described below.

Control Sections (CSECT) and External References (EXTRN/WXTRN). This format is as follows:



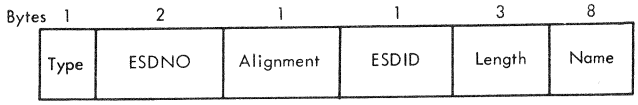
Entry Definitions. This format is as follows:



Flag - Set to 1 to indicate completion of the item.

Label definition ID - External symbol dictionary ID of the containing control section.

External Dummy References (ENTRY). This format is as follows:

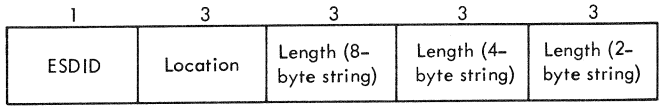


ESDNO - Used to refer to the DSECT if this item was generated by a Q-type address reference to a DSECT. It is zero if the item was generated by a DXD instruction.

Alignment - One less than the number of bytes in the unit of alignment, e.g., 7 for double word alignment.

Literal Base Table

This format is as follows:



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

Location - This is the relative address obtained from the statement work bucket 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.

PHASE ORGANIZATION

Phase F7 is organized as follows:

1. Mainline control - IEUF7C.
2. Functional routines.
   a. GET statement - IEUF7X.
   b. DC, DS evaluation - IEUF7D.
   c. External symbol dictionary processor - IEUF7E.
   d. TESTRAN processor - IEUF7N.

3. Common Subroutines (common to routines within Phase F7).
   a. Symbol table function - IEUF7S.
   b. Expression evaluation - IEUF7V.
   c. Error logging function - IEUF7L.
   d. Literal DC generator - IEUF7G.
   e. I/O Subroutines.

IEUF7C - MAIN LINE CONTROL (FLOWCHARTS 13-18)

Control is passed to Phase F7 Mainline Control (IEUF7C) by the Phase F7 initialization routine, IEUF7I.

Program modifications are made based on the settings of the cross-reference and TESTRAN option bits. IEUF7N is called by IEUF7C to generate TESTRAN cards for all edited statements if the option bit is set.

IEUF7X is called to move the next edited text record into the text work area after putting the current (processed) text record onto the output file.

The mode is tested (program modification). If substitution has been made, control is passed to substitution control.

If the type of operation is assembler OP, control is passed to assembler OP control.

Machine operations are processed, and the location counter incremented by the operation length. If an external symbol dictionary identification has been assigned, control is passed to the external symbol dictionary routine to initiate private code.

If a name is present, the symbol table is tested for possible overflow. If the symbol table is full, the mode is changed to substitution, and control passed to substitution control.

If a literal is referenced in the operand field, control is passed to IEUF7D to make a literal entry in the symbol table.

If the cross-reference bit is set, the operand is examined for references to symbols.

The operation length is then calculated from the hex code.

Assembler operation codes are processed by examining the assembler switch code for the following:

- Uninitiated private code (if the external symbol dictionary ID = 0).
- Possible symbol table overflow.
- Location counter reference.
- Special cross-reference scan to generate cross-reference records.

The internal hex code for assembler operation codes is used to compute a branch address to the specialized assembler operation routine.

In the substitution mode, IEUF7C tests substitution is required, a work bucket is attached for each symbol referenced in the operand.

For machine operation codes, IEUF7C tests name fields for multiply defined symbols and evaluates literals for duplicates.

For assembler operation codes, IEUF7C processes assembler operations in the substitution mode only when substitution is required.

IEUF7X - PHASE F7 GET STATEMENT ROUTINE (FLOWCHART 19)

IEUF7X is used by IEUF7C to move edited text records into the text work area and put them back into the text stream.

On all but the first time called, IEUF7X puts the current text record onto the text file by calling PUTXT.

If an error record is in the build area, IEUF7X puts the error record onto the next record following the text record and clears the error record in core switch.

If the literal switch is set, the next record will be moved into the text work area from the literal entry in the symbol table. Otherwise, IEUF7X calls GETPT for a pointer to the next text record in the input buffer.

If an end-of-file is detected, IEUF7X moves a QUIT record into the text work area if the mode is assignment mode. If the mode is substitution, the file is TCLOSEd and re-opened beginning with the first text record.

IEUF7X tests each text record for a record type if edited. If the record is not edited, it is put out on the output text file, and the next record is examined.

Edited text records are moved into the text work area for processing by Phase F7.

Edited-generated records are converted to suitable format for Phase F7 processing. The hex code is set from the operation code conversion table, or -- if not found there -- from the OPSYN chain of the symbol table, and substituted fields are adjusted for leading and trailing blanks.

IEUF7X sets absolute pointers to the operand field and appends fixed field and symbol work buckets (if any exists).

The operand field on machine operations is scanned for literals. If an equal sign is found outside of quotes, the literal in the operand indicator is set, and the literal work bucket is appended to the text record.

Fields are tested for legality as follows:

- The name field is tested for legal characters, too many characters, and leading alpha character.
- Assembler operations are tested for name field required or not allowed and operand field required or not allowed.

IEUF7D - DC/DS EVALUATION ROUTINE (FLOWCHART 20)

This routine is called by IEUF7C to process DCs, DSs, DXDs, literals, and literal DCs. A complete syntax check is done for all DC types, and appropriate error messages are logged when necessary. One 15-byte DC work bucket is attached to the appended fixed field of the text record for each DC, DS, DXD, and literal DC operand for use by the Phase F8 DC evaluation routine (IEUF8D). In the external symbol dictionary, a table entry is made for each valid constant in a V-type or Q-type DC.

One complete statement is processed in each pass, and control is returned to IEUF7C by executing an unconditional branch to location CTRTRN (ACT).

IEUF7E - EXTERNAL SYMBOL DICTIONARY PROCESSOR ROUTINE (FLOWCHARTS 21-23)

This processor is called whenever any of the assembler operations COM, START, CSECT, DSECT, DXD, ENTRY, EXTRN, WXTRN, or ORG or a V- or Q-type address constant is encountered. It is also called at the beginning and end of assembled code. Three other entry points are used for ENTRY, EXTRN, and control sections in the substitution mode.

The functions performed are as follows:

- Generating external symbol dictionary entries.
- Updating the location counter in external symbol dictionary entries.
- Making symbol table entries for names in the statements handled.
- Setting and maintaining the control table switches CBDNO, CBDSW, CCMNO, CESDID, CESDNO, CNOESD, CPCNO, CTPCSW, CSTVAL, CSGCTR, CLASID, CTNDID, CTCMSW, CTESDP, CTESRN, CTESRP, CTFSTN, CTLOC, and CTYPE.
- Issuing various error messages.

IEUF7N - F7 TESTRAN ROUTINE

If the TESTRAN option is exercised, control is passed from IEUF7C to IEUF7N after the process of each statement that defines a symbol or affects the location counter in any way.

The output records (cards) of IEUF7N are written on SYSPUNCH and/or SYSGO data sets. Subsequent executions of the object program in the TESTRAN mode use this information.

IEUF7S - SYMBOL TABLE SUBROUTINE (FLOWCHART 24)

(This routine is resident in module IEURTA). The symbol table processor has three entry points in Phase F7: STPUTR, STGETR, and STROOM. Their function is to put symbols into the table, retrieve symbols from the table, and test whether room exists for another symbol.

The symbol table and the external symbol dictionary share an area of main storage. The symbol table starts at the low-numbered end, and the external symbol dictionary starts at the high-numbered end. Room must be left by the symbol table for two external symbol dictionary segments of 260 bytes each (16 items 16 bytes long, plus 4 bytes for overflow addressing). Apart from this restriction, overflow does not occur until the external symbol dictionary and the symbol table are about to meet.

STPUTR tests to see whether a duplicate exists and, if not, puts the given symbol into the table with its attributes. A type 1 or 3 cross-reference is also made. (See "Cross-Reference Record Format.")

STGETR tests whether the requested symbol is in the table and, if so, gives the address of the first byte in the entry after the name field. If not, zero is returned.

STROOM determines whether overflow can occur on the next STPUT. If so, then if room can be made for the symbol table by so doing, or if the external symbol dictionary processor made the call, the external symbol dictionary is put on the overflow file. If not, the need to enter substitution mode is signalled.

IEUF7V - EXPRESSION EVALUATION ROUTINE (FLOWCHART 25)

SP1 at entry contains a pointer to the first character of the expression. SP1 at exit contains a pointer to the character which caused IEUF7V to terminate. The terminating character will always be a left or right parenthesis, blank, or comma, unless there was a syntactical error, in which case SP1 will be zero.

SP2 at exit contains the result, if the expression is absolute. If the expression is relocatable, SP2 contains a pointer to a full word value followed by the RLIST. If the expression contains an error, SP2 is zero.

Upon exit from IEUF7V, the condition code has the following meaning:

Code	Meaning
0	absolute expression
1	simple relocatable expression
2	complex relocatable expression
3	evaluation impossible (error)

Syntax errors cause immediate exit from IEUF7V; errors other than syntax are logged, and normal processing is continued.

IEUF7V has the following functions:

- Evaluate expressions.
- Log type 2 cross-references (XREF) - (only IEUF8V).
- Convert self-defining values (SDVCF).
- Detects the following errors and passes the information to IEUF7L:
  Relocatability error
  Self-defining value too large
  Arithmetic overflow
  Symbol not found - (IEUF8V)
  Symbol not previously defined - (IEUF7V)
  Two terms not separate
  Illegal character
  Too many terms
  Two operators illegally coupled
  Too many levels of parentheses
  Expression end premature
  Invalid symbol
  Expression value too large

Algorithm description: A term is a relocatable or absolute symbol, a length attribute reference (L'sym), location counter reference (*), or a self-defining value. When a term is encountered, its value is entered in the next available position in the TERMS list. If it is a relocatable term, the sign code and its external symbol dictionary ID are entered in the next available position in the RLIST list. If it is an absolute term, the RLIST pointer is lumped to the next half-word, in effect putting zeros into that position, since all the tables are zeroed during IEUF7V initialization.

Type 2 cross-references are made during Phase F7 assignment mode. (See "Cross-Reference Record Format.")

When an operator is encountered, its code is entered in the next available position in the OPRNS list, providing its hierarchy is greater than the previous operator. Hierarchy codes are as follows:

0	(), b
1	+
2	-
3	*
4	/

The code 0 for a left parenthesis is always entered in the OPRNS list. A right parenthesis forces all operators in the OPRNS list back to the preceding left parenthesis. A blank, comma, or a left or right parenthesis which legitimately ends the expression forces all operators in OPRNS to the top of the list. An operator with a code less than or equal to the previous code forces only the previous operator.

When an operator is forced, the arithmetic is performed between the last two entries in the TERMS list, and the result is stored in the position of the first entry involved in the arithmetic. Also, the sum of the two corresponding NTRMS entries is placed in the position of the first entry.

COND is initially set to 0 on entry to IEUF7V. It is then set to 1 each time a right parenthesis, +, or - is encountered; to 2 each time an * or / is encountered; to 3 each time an absolute term or left parenthesis is encountered; and to 4 each time a relocatable term is encountered. The COND switch setting determines the validity of an operation. See Table 11. For example, / is valid only when COND=3 (/ follows an absolute term or left paren.). If COND=4, a relocatability error is logged (/ follows a relocatable term). A syntax error is logged if COND=0 (expression begins with /) or if COND=1 or 2 (/ follows +, -, *, /, or right paren.).

Work tables used in IEUF7V are as follows:

TERMS (16 full words) - Entry is made for each term in the expression. At end of evaluation, the first full word location contains the final result; subsequent table entries contain intermediate results.

OPRNS (20 bytes) - Entry made for each operator in the expression.

NTRMS (16 bytes) - A 2 is entered for each term. At end of evaluation, the first byte contains a value equal to twice the number of terms in the expression.

RLIST (16 half-words) - Sign code (1 for +, 2 for -) and external symbol dictionary ID are entered for each relocatable term; zeros appear for each absolute term and undefined symbol. During addition and subtraction, RLIST entries are zeroed when the signs are opposite and the external symbol dictionary IDs are the same.

Upon exit, a simple relocatable expression will have a + sign and the external symbol dictionary ID of its impaired positive term in the first half-word, and the remaining 15 half-words will contain zeros. A complex relocatable expression at exit will have a non-zero half word for each unpaired relative term. The non-zero half words will be scattered in the table. A complex relocatable expression may also be the result of a single unpaired negative term. A minus sign (-) and the external symbol dictionary ID of this term will appear in the RLIST.

Table 11. Condition Switch Settings

Character Encounterd	Previous Setting	Action	Flowchart References
start	--	set COND = 0	
(	0/1/2	set COND = 1	LPAR
	3/4	if PCNTR > 4, log error IEU026	
)	0/1/2	log error IEU085	RPAR
	3/4	set COND = 3 if PCNTR = 0, end of expression	
+ or -	3/4	set COND = 1	LTCOM
	0/1/2	log error IEU085	LOOP
*	0/1	set COND = 4 (* is location counter ref.)	LTCOM
	2	log error IEU085	
	3	set COND = 2 (* is mult.)	
	4	log error IEU025	
/	3	set COND = 2	LTCOM
	0/1/2	log error IEU085	LOOP
	4	log error IEU025	
absolute term	3/4	log error IEU085	Term Computing Routines
	0/1/2	set COND = 3	
relocatable term	0/1	set COND = 4	
	2	log error IEU025	
	3/4	log error IEU085	
, or blank	0/1/2	log error IEU039	BLCOM
	3/4	if PCNTR > 0, log error IEU039	

IEUF7L - ERROR LOGGING FOR PHASE F7 AND F8 (FLOWCHART 26)

(This routine is resident in module IEURTA). This routine is called to attach error messages to an edited text record.

IEUF7L tests the error switch in the control table to determine if an error record for the current text record is in the error record build area. If there is, the error count is compared with the maximum allowable number of errors (16). If the count is equal to the maximum, the current and all subsequent error messages are ignored.

If there is no error record in the build area, IEUF7L tests the "error record follows" bit in the text record. If the bit is set, the error record is moved into the build area from the text file. If not, the error record is initialized in the build area, and the "error record follows" bit (TXERI) is set in the text record. In either case, the error switch in the control table is set for subsequent calls on the current text record.

The error message is added to the error record, and the error count is incremented by one.

The relative column pointer is added to the error message. If no column pointer is required, it is set to zero.

Control is then returned to the calling routine.

IEUF7G - LITERAL DC GENERATOR (FLOWCHART 27)

IEUF7G is a routine which builds a literal DC edited text record for an outstanding literal entry in the symbol table. IEUF7X invokes IEUF7G once for each literal DC that is to be built into the edited text. IEUF7G then moves in pertinent information such as the record length, record type X'60' for edited generation, operation type X'80' for assembler, operation code X'25' for literal, and operation and name field lengths of zero. After the text has been generated, control is returned to the caller.

IEUF7I - PHASE F7 INITIALIZATION AND I/O INITIALIZATION

The I/O portion of phase initialization OPENs the three utility files. BSAM logic including READ, WRITE, CHECK, and TCLOSE are employed for all data sets. QSAM logic (PUT) is used for the SYSGO and SYSPUNCH data sets. The routine initiates a READ of the first block of the text stream and initializes text, literal base table, and cross-reference pointers.

F7IO functions for PHCLS, GETPT, GETXTM, PUTXT, CLSTXT, CWRESD, CRDESD, PUTXRF, PUTLBT, and SYSO are described below.

The root segment transfers control to F7 which in turn transfers to IEUF7C.

PHCLS - Phase Close

The I/O portion of the phase close function closes SYSUT1, SYSUT2, and SYSUT3 and inserts the following parameters into the I/O portion of the assembler control table:

CTXUIN (1 Byte) - Data file designator for the current input text.

CTOUT2 (1 byte) - Data file designator for the alternate overflow file (other than SYSUT3).

CTRLBT (4 bytes) - Pointer to the first literal base table block on the overflow file, SYSUT3.

CTCLBT (2 bytes) - Count of the number of literal base table physical blocks which have been written onto SYSUT3.

CTRXRF (4 bytes) - Pointer to the first cross-reference table block contained on the overflow file, SYSUT3.

CTCXRF (2 bytes) - Count of the number of cross-reference physical blocks which have been written onto SYSUT3.

CTONWP (8 bytes) - Pointer to the next sequential write position on the overflow file, SYSUT3.

Prior to TCLOSEing SYSUT3, the literal base table pointers are tested to determine if any literal base table entries have been made. If so, an end-of-file label is embedded into the literal base table stream, and the partially filled block is written onto the overflow file, SYSUT3. If no literal base table entries have been made, then CTCLBT is cleared to zeros.

PHCLS may be called by an unconditional branch to PHCLS. The routing ultimately transfers control to the root segment for calling the next phase.

GETPT - Get Point

GETPT points to the next logical text record within the input text stream.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

```
Entrance Procedure:  L    SRB,CTXTIO(ACT)
                     BAL  SRR,GETPT(SRB)
                     (Normal return)
```

Output Parameters:

SP1=FBA	First byte address of the first segment of a logical text record.
=0	End of file was read, the text stream has terminated.
SP2=FBA	First byte address of the second segment of a "broken" logical text record.
=0	Text record was not broken, consequently no second segment.

Method. GETPT double-buffers the text stream. Logical buffers may be split between two physical blocks. The routine points to each record in sequence. The pointer is pointing to the most significant byte of the record length indicator (RLI).

Restrictions and Assumptions. The "last record bit" contained within FLAGA must be set to terminate the input text stream. A logical record must be contained within two physical blocks.

GETXTM - Get Text and Move

GETXTM transfers a logical record from the input text stream to an area specified by the user.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1=FBA	First byte address of the area to which the record is to be transferred.

```
Entrance Procedure:  L    SRB,CTXTIO(ACT)
                     BAL  SRR,GETXTM(SRB)
                     (Normal Return)
```

Output Parameters:

SP1=FBA	First byte address of user's work area (same as on input).
=0	End-of-file was read; the text stream has terminated.

Method. Upon entering, GETXTM tests a global switch which is set by the GETPT subroutine. If the switch is set, GETXTM clears the switch and transfers the record that is currently being pointed. If the switch is not set, GETXTM calls GETPT to point to the next logical record in the input text stream. GETXTM then transfers that record to the user's work area. Thus, if the user wishes the next logical record moved to his work area, he can call GETXTM without first calling GETPT.

If the text record is segmented, the routine joins the two segments together to produce a single contiguous record. In so doing, it drops the second record length indicator and updates the first record length indicator to the total record byte count. In addition, the "break flag bit" contained within FLAGA is cleared to zero.

PUTXT - Put Text

PUTXT retrieves a logical record from the input buffers or from an area specified by the user, and transfers the record to the output buffers for eventual output to a utility data set.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1=FBA	First byte address of the text record contained in the user's work area.
=0	A flag to indicate the record to be PUT is contained in a text input buffer.

```
Entrance Procedure:  L    SRB,CTXTIO(ACT)
                     BAL  SRR,PUTXT(SRB)
                     (Normal return)
```

Output Parameters:

SP1=FBA_1	First byte address of the text record contained in the user's work area (same as input).

=FBA_2 First byte address of the text record contained in the output buffer if record was transferred from the input buffer.

=0 End-of-file was read while trying to PUT the next logical record from the input text stream.

Method. If SP1 is zero, PUTXT sets a global switch and calls GETXTM. GETXTM tests the switch and, if set, transfers the text record from the input buffer to the output buffer. If the GETPT global switch is not set, GETXTM will call GETPT to point to the next logical record. Thus, the next logical record in the text stream can be transferred directly to the output buffer by simply calling PUTXT without first calling GETPT or GETXTM.

If SP1 is non-zero, the record is transferred from the area specified by SP1 into the output buffer.

Restrictions and Assumptions. A logical record must not exceed in length one physical output block.

CLSTXT - Close Text

CLSTXT TCLOSEs the input text file, embeds an end-of-file label into the output text stream, and TCLOSEs the output text file. In addition, it interchanges the utility file designators such that the current input text file becomes the future output text file, and the current output text file becomes the future input text file.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

```
Entrance Procedure:  L     SRB,CTXTIO(ACT)
                     BAL   SRR,CLSTXT(SRB)
                     (normal return)
```

Output Parameters: None

Method. The output buffer management subroutine is called, an end-of-file label is embedded into the output text stream, and the output file is TCLOSEd. The I/O designators are interchanged, and the text buffer pointers are initialized. This routine is called by PHCLS. Hence, unless the phase anticipates an iteration on the text stream, this routine should not be called by mainline control.

CWRESD - Write External Symbol Dictionary

CWRESD writes the external symbol dictionary (ESD) onto the overflow file, SYSUT3, and NOTEs its position for future reference.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1=FBA First byte address of the external symbol dictionary.

```
Entrance Procedure:  L     SRB,CTXTIO(ACT)
                     BAL   SRR,CWRESD(SRB)
                     (Normal return)
```

Output Parameters:

SP1=NOTEd record position on the overflow file.

Method. The routine tests the write positioning required switch. If set, the file is POINTed to the next write position. If not set, the file is assumed positioned for the next write. The external symbol dictionary block is then written onto the overflow file and its position NOTEd. The note label is passed on to the user for future reference.

Restrictions and Assumptions. The routine does not keep count of the number of external symbol dictionary blocks read or written. It assumes the user is requesting a block which has been previously written.

CRDESD - Read External Symbol Dictionary

CRDESD POINTs the overflow file to the requested external symbol dictionary and reads it into an area specified by the caller.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1=L A NOTEd record position used to POINT to the desired external symbol dictionary segment.

SP2=FBA First byte address of user's input area.

```
Entrance Procedure:  L     SRB,CTXTIO(ACT)
                     BAL   SRR,CRDESD(SRB)
                     (Normal return)
```

Output Parameters: None

Method. Using SP1, the overflow file is POINTed to the requested external symbol dictionary, the external symbol dictionary is read into the area specified by SP2, and the write positioning required switch is set.

Restrictions and Assumptions. The routine does not keep count of the number of external symbol dictionary blocks read or written. It assumes the user is requesting a block which has been previously written.

PUTXRF - Put Cross-Reference

PUTXRF points to the next available area in the cross-reference output area for building a cross-reference logical record.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

Entrance Procedure:
```
L     SRB,CTXTIO(ACT)
BAL   SRR,PUTXRF(SRB)
(Normal return)
```

Output Parameters:

SP1=FBA — First byte address of the next available record location contained within the cross-reference (XRF) buffer.

Method. The routine is called each time a cross-reference record is to be built. The routine advances through the buffer at 17-byte increments until the block is filled. At this point, the buffer is written onto the overflow file at the next available write position. The first block written onto the overflow file is NOTEd for future reference.

Restrictions and Assumptions. The routine assumes a record will be built at the POINTed location since it merely advances to the next 17-byte location each time it is called.

PUTLBT - Put Literal Base Table

PUTLBT points to the next available area in the literal base table output area for building a literal base table logical record.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

Entrance Procedure:
```
L     SRB,CTXTIO(ACT)
BAL   SRR,PUTLBT(SRB)
(Normal return)
```

Output Parameters:

SP1=FBA — First byte address of the next available record location contained within the literal base table buffer.

Method. The routine is called each time a literal base table record is to be built. The routine advances through the buffer at 13-byte increments until the block is filled. At this time the buffer is written onto the overflow file at the next available write position. The first block written onto the overflow file is NOTEd for future reference.

Restrictions and Assumptions. The routine assumes that a record will be built each time it is called since it merely advances to the next 13-byte location.

SYSO - System Output

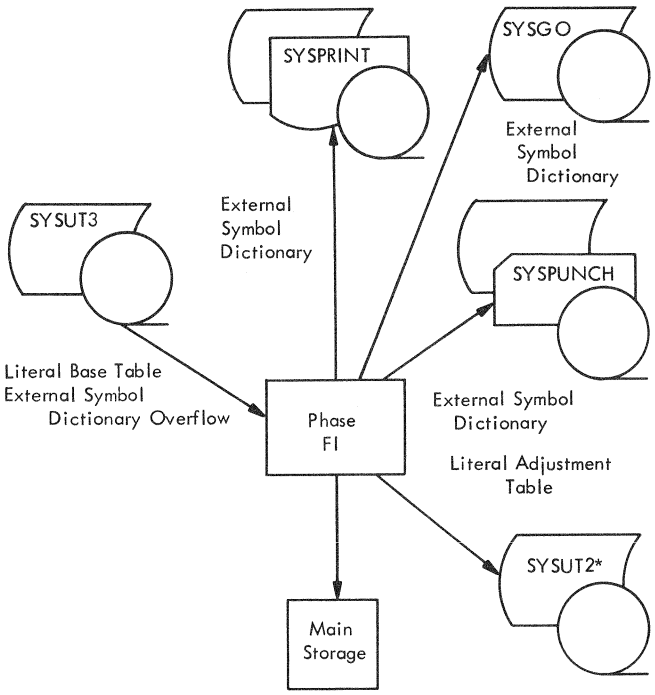
SYSO outputs 80-character logical records to either the SYSPUNCH or SYSGO data sets or to both. On entry, SP1 points to the first byte of an 81-character buffer where the first character is an internal control character. SYSO tests the SYSPUNCH and the SYSGO option bits. If either one is set, the contents of the 81-character buffer (except for the control character) are transferred accordingly to the SYSPUNCH or SYSGO data sets. If both are set, the contents of the buffer are transferred to both data sets.

OVERALL OPERATION (FLOWCHART 28)

The main function of Phase FI (F Interlude) is to write the external symbol dictionary on the SYSPRINT, SYSPUNCH, and/or SYSGO data sets. External symbol dictionary segments and literal pool bases are located either in main storage or in an overflow segment on SYSUT1.

I/O FUNCTIONS

Phase FI processes literal base table and external symbol dictionary entries generated in Phase F7 and outputs literal adjustment table entries. See Figure 12. The literal base table and literal adjustment table are passed at the GET/PUT logical record level, and the external symbol dictionary input is called at the READ level. The I/O subroutines interface with the operating system at the READ/WRITE level only, performing their own blocking and deblocking functions for the logical record entries. The system output routines SYSO and SYSL are called to output external symbol dictionary data.



*If symbol table overflow occurred in Phase F7, SYSUT1 may be used instead of SYSUT2.

Figure 12. I/O Flow for Phase FI

LITERAL ADJUSTMENT TABLE FORMAT

The format for the literal adjustment table is as follows:

Bytes	1	3	1	3	1	3	1	3
	ESD ID	A	ESD ID	B	ESD ID	C	ESD ID	D

ESD/ID — External symbol dictionary identification of the 3 bytes (A) 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 A).

B — Same as A, except as applicable to the 4-byte string.

ESD/ID — Same as previously described (for A).

C — Same as A, except as applicable to the 2-byte string.

ESD/ID — Same as previously described (for A).

D — Same as A, except as applicable to the 1-byte string.

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

Trailer — Indicates the end of the literal adjustment table. This format is as follows:

Bytes	1	1	1	1
	7 F	7 F	7 F	7 F

I/O SUBROUTINES

FII — FI Initialization

The I/O portion of phase initialization determines if any literal base table entries had been output from Phase F7. If so, the first literal base table block is read from the overflow file, SYSUT1, and the pointers to the first logical record entry

within the block are initialized. Control is then transferred to the mainline control driver.

Phase IEUF7I (the Phase F7 I/O routine) transfers control to FII which in turn transfers to Phase FI mainline control.

FICLS — FI Phase Close

The I/O portion of the phase close function embeds an end-of-file label into the literal adjustment table output stream and writes the last literal adjustment table block onto the alternate overflow file. SYSUT1 and the alternate overflow file are TCLOSEd, and the following entry is placed into the assembler control table:

CTCLAT — Count of the number of physical literal adjustment table blocks which have been written onto the alternate overflow file.

Control is then transferred to the next phase.

FICLS should be called only after mainline control has finished its phase. FICLS may be called by an unconditional branch to FICLS.

GETLBT — Get Literal Base Table

GETLBT points to the next logical record entry contained within the literal base table input area.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

Entrance Procedure: BAL SRR,GETLBT
(Normal return)

Output Parameters:

SP1 = FBA — First byte address of next logical record in the literal base table input buffer.
= 0 — End-of-file was detected; there are no more literal base table records.

Method. The subroutine is called each time a literal base table logical record is required. The routine advances through the buffer at 13-byte increments until the entire block has been processed, at which time the next literal base table physical block is read from the overflow file. If either an end-of-file label is detected or the block count becomes zero (whichever appears first), the end-of-file flag is set, and control is passed to the caller.

RDESD — Read External Symbol Dictionary

RDESD POINTs the overflow file to the requested external symbol dictionary and reads the block into an area specified by the caller. RDESD is embedded into the section LPFI2. LPFI2 reads the entire external symbol dictionary into core, segment by segment. Logic is identical with Phase F7 RDESD.

PUTLAT — Put Literal Adjustment Table

PUTLAT points to the next available area in the literal adjustment table output area for building a literal adjustment table logical record.

Calling Sequence. The calling sequence is as follows:

Input Parameters: None

Entrance Procedure: BAL SRR,PUTLAT
(Normal return)

Output Parameters:

SP1 = FBA — First byte address of the next available record location contained within the literal adjustment table output buffer.

Method. The routine is called each time a literal adjustment table logical record is to be built. The routine advances through the buffer at 16-byte increments until the block is filled, at which time the buffer is written onto the alternate overflow file. The alternate overflow file is designated as that file which is neither the prime overflow file, SYSUT3, nor the current text file. Which file is designated is contingent on the number of passes through the text stream which were executed by the previous phase, Phase F7.

SYSL — System List

SYSL outputs a 121-character line to the system list data set, SYSPRINT.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1 = FBA — First byte address of a 121-character formatted print line. The first character is the internal control

character. The numeric value of the first character dictates the number of lines to be spaced prior to listing the formatted line. Zero is single spacing, 63 or greater is page eject.

Entrance Procedure: BAL SRR,SYSL

Output Parameters: None.

Method. The SYSL option bit is tested. If set, the line is written on SYSPRINT. If not set, a simple return is executed. The user should not concern himself as to whether the line was written or not.

SYSO — System Output

This subroutine outputs an the contents of the 81-character buffer (except for the control character) to either the SYSPUNCH or the SYSGO data sets.

Calling Sequence. The calling sequence is as follows:

Input Parameters:

SP1 = FBA First byte address of an 81-character buffer where the first character is an internal control character.

Entrance Procedure: BAL SRR,SYSO

Output Parameters: None

Method. SYSO tests the SYSPUNCH and the SYSGO option bits. If either is set, the contents of the 81-character buffer (except for the control character) are transferred to the data sets indicated.

MAIN LINE CONTROL

If the external symbol dictionary is not in core, it is fetched one segment at a time using the segment residence table. The adjustment table is constructed from control sections on the external symbol dictionary by accumulating their lengths and aligning to the next higher double word.

Another pass through the external symbol dictionary outputs the listing and cards item by item. These consist of name, type, ID, address, length, and LDID, as appropriate, on the listing; and name, type, address alignment, and length or LDID on the cards, with one ID per card to identify its first SD, PC, ER, WX, CM, or XD type.

The adjustment table built in FI3 and the literal base table are used to build the literal adjustment table.

The adjustment table is resident in module IEURTA and overlays IEUF7S and IEUF7L.

OVERALL OPERATION (FLOWCHARTS 29-36)

Phase F8 makes the final pass through the program text, which is read from SYSUT1 or SYSUT2, depending on the number of iterations. During this pass, any operands which were not processed by Phase F7 are processed from the last symbol table created during that phase.

Any self-defining values which were not converted to their binary values are now processed. All address expressions are evaluated, and the results are substituted for the expressions. Addresses are restructured into a base register and displacement format.

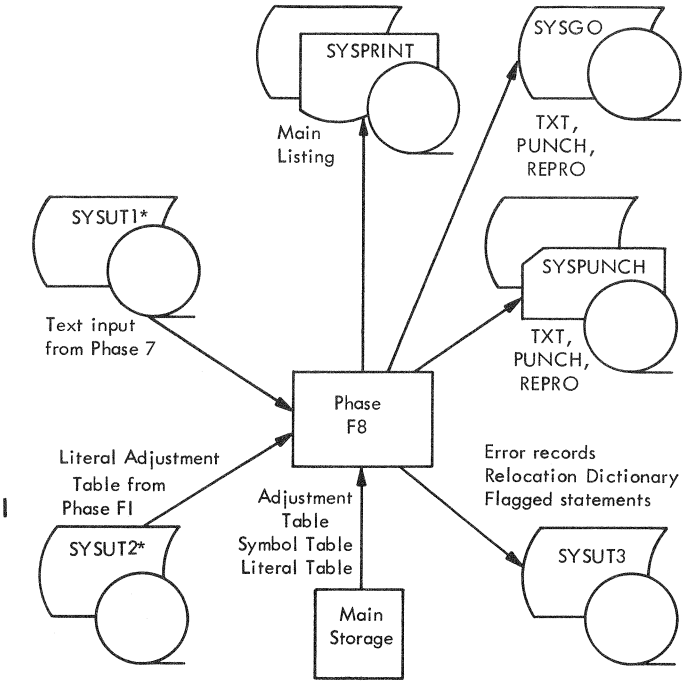
At the same time, the completely assembled text is written in relocatable object program format on SYSPUNCH or SYSGO, and in program listing format on SYSPRINT.

Invalid statements are flagged on SYSPRINT. Error records together with the statements flagged are created and written on the overflow file, SYSUT3, to be listed by the Post-Processor Phase.

The relocation dictionary is built during Phase F8, and segments can overflow onto SYSUT3.

I/O FUNCTIONS

Phase F8 passes through the text stream once scanning the logical text records which were output from Phase F7. See Figure 13. The phase inputs the literal adjustment table blocks from the alternate overflow file and outputs cross-reference, relocation dictionary, and diagnostic records with flagged statements onto the overflow file for subsequent processing by the Post-Processor Phase. The cross-reference and relocation dictionary records are output at the PUT level where the cross-reference records are inserted into the cross-reference output stream from where they left off in Phase F7. Error records are output at the WRITE level. In addition, records are output to the SYSPRINT, SYSPUNCH, and SYSGO files via the output subroutines SYSL and SYSO.



●Figure 13. I/O Flow for Phase F8

RELOCATION DICTIONARY ENTRY FORMAT

This format is as follows:

Bytes	1	1	1	1	3
	Table ID	Position ESD/ID	Relocation ESD/ID	Flag	Symbol address

Table ID -- Each group of 20 RLD entries is preceded by a 1-byte table identifier of '08'.

Postion ESD/ID — Number of the control section where the address constant is located.

Relocation ESD/ID — Number of the control section where the symbol is defined.

Flag —

Bits 0-1	00
Bits 2-3	00 - A-type and Y-type address constants and CCW. 01 - V-type address constant. 10 - Q-type address constant. 11 - CXD.
Bits 4-5	Length of address constant minus one (L-1).
Bit 6	External symbol dictionary (ESD) ID sign. 0 — plus (+) 1 — minus (-)
Bit 7	0 — next entry on the same card has the same position ID and the same relocation ID. 1 — next entry on the same card has a different position ID and/or relocation ID.

NOTE: There is no carry-over from card to card. That is, the last entry on a card always has a 1 in bit position 7 even if the first entry on the next card has identical position ID and relocation ID fields.

Symbol address - Assembler assigned address of a symbol used in A-, Y-, or V-type address constants, or of the second operand of CCW.

PHASE ORGANIZATION

The following control sections comprise Phase F8: IEUF8I, IEUF8C, IEUF8M, IEUF8A, IEUF8P, IEUF8D, IEUF8V, IEUF8L, IEUF8N, and IEUF8S. These are self-contained routines which may be link-edited separately. Communication between routines is via registers and the ACT table. When IEUFI passes control to Phase F8, the above routines are loaded into core and remain there until the completion of Phase F8 execution.

IEUF8I -- PHASE F8 INITIALIZATION AND I/O

On entry to Phase F8, the overflow file, SYSUT3, is positioned for writing the relocation dictionary and diagnostic records. If any literal adjustment table records were written by Phase FI, the first literal adjustment table block is read from the alternate overflow file. The literal adjustment table pointers are initialized to point to the first logical record contained within the literal adjustment table input area. GETMAIN for a buffer to contain flagged statements is issued, if the TERM option is in effect. The relocation dictionary type indicator is inserted into the first byte of the relocation dictionary output buffer (the cross-reference output buffer was initialized in Phase F7), and control is transferred to Phase F8 mainline control, IEUF8C.

IEUF8I SUBROUTINES

GETXTM -- Get Text and Move

GETXTM retrieves the next logical record from the input text stream and moves the record to an area specified by the user. Each time the routine is entered, a pointer is advanced to the next logical record in the input text stream. The record is moved from the input buffers to an area specified by SP1. The input stream is double-buffered to increase input speed and processing efficiency. If an end-of-file label is encountered, the EOF flag is set and passed to the caller. GETXTM sets a switch which prevents any further processing of the text stream.

GETLAT -- Get Literal Adjustment Table

GETLAT points to the next literal adjustment table logical record contained within the literal adjustment table input area. The routine is called each time a new literal adjustment table record is desired. On exit, SP1 contains the first byte address of the next literal adjustment table logical record. The routine advances through the buffer at 16-byte increments until the block is empty, at which time the next block is read from the alternate overflow file. SP1 returns with 0 if there are no more literal adjustment table entries.

PUTRLD -- Put Relocation Dictionary

PUTRLD points to the next available area in the relocation dictionary output area for building a relocation dictionary logical record. The routine is called each time a relocation dictionary is to be built. SP1 returns with the first byte address of the next available record location in the relocation dictionary output buffer. The routine advances through the buffer at 6-byte increments until the block is filled, at which time the buffer is written onto the overflow file, SYSUT3. The first block written onto the overflow file is NOTEd for future reference.

PUTXT -- Collect Flagged Statements

If the TERM option is in effect PUTXT collects the flagged statements and outputs them on SYSUT3. The routine is called each time a source record is moved to the SYSPRINT buffer. The routine puts the record in a buffer set aside for this purpose (see Figure 16). When an error is found or the buffer has been filled, and the last source record moved into the buffer contains a continuation character, the buffer is written on SYSUT3. For a valid statement exceeding six records a POINT is made. Other valid statements are not written on SYSUT3.

WTERR -- Write Error Message

WTERR outputs an error record onto the overflow file for eventual listing by the Post Processor Phase, Phase FPP.

WTERR is called with SP1 pointing to the first byte of an error record.

SYSL -- System List

SYSL outputs a 120-character line to the SYSPRINT file. On entry, SP1 points to the byte of a 121-character formatted printer line. The first character is the carriage control character. The numeric value of the first character dictates the number of lines to be spaced prior to listing the formatted line. Zero is single spacing, 63 or greater is page eject. The SYSPRINT option bit is tested. If set, the line is transferred to the SYSPRINT data set. If not set, a simple return is executed.

SYSO -- System Output

(See the description of SYSO under IEUF71 in Phase F7.)

PHCLS -- Phase F8 Close

The phase close subroutine TCLOSEs the text file, embeds end-of-file labels into the relocation dictionary output stream, and writes their buffers onto the overflow file. The following nine parameters are passed to the Post Processor Phase through the assembler control table:

CTPCHI (1 byte) - Option bits for DECK, LOAD, RENT, LIST, ALGN, DOS, XREF (in that order).

CTTRMI (1 byte) - Option bits for TEST, TERM, NOM, STMT .

CTCXRF (2 bytes) - Count of XREF blocks contained on the overflow file.

CTCRLD(2 bytes) - Count of relocation dictionary blocks contained on the overflow file.

CTCERR (2 bytes) - Count of the number of error messages contained on the overflow file.

CTRXRF (4 bytes) - Location of the first XREF block located on the overflow file.

CTRRLD (4 bytes) - Location of the first relocation dictionary block located on the overflow file.

CTRERR (4 bytes) - Location of the first diagnostic message located on the overflow file.

CESDNO (2 bytes) - Deck number.

CTITLE (4 bytes) - Deck identification.

PHCLS transfers control to IEUFPP, the Post Processor Phase.

IEUF8C - MAIN LINE CONTROL (FLOWCHART 29)

Control is passed to IEUF8C for each record; IEUF8C in turn passes control to the various routines necessary to process that particular record. Program sequences within the mainline control that perform specific functions are described below.

IEUF8C SUBROUTINES

ERLOD8

Errors encountered in Phase F8 for a record are appended to the corresponding Phase F7 error record and are put in the record work area F8WORK (ACT).

ENDOFF

ENDOFF processes any error encountered in the final record and exits to the IEUF8I phase close routine.

SETWBP

CTXWBP(ACT) points at the first symbol work bucket in the input record. If there are no symbol work buckets, CTXWBP(ACT) will be set to zero. CTXABP(ACT) is set to point at the appended fixed field of the input record.

SRLIGN

SRLIGN makes alignments as necessary on all machine ops, literal DC, DC, DS, CCW, and CNOP. TXALIN in F8WORK(ACT) is investigated. If this 3-bit designator is non-zero, one to seven alignment bytes are output.

IEUF8M - MACHINE OPERATION PROCESSOR (FLOWCHART 30)

IEUF8M processes the operand field of all machine ops. Decomposition, adjustment, and formating occurs in this routine. The decomposition routine Using table is shown in Figure 14. IEUF8M is entered from the IEUF8C routine. FRB (Register 12) is loaded with the base address of IEUF8M. Control is passed to the assembler control table where registers 4, 5, 6, and 7 are saved. Control is then passed to the address in FRB.

IEUF8M SUBROUTINES

RR1, RR2, RR3, RR4, RX1, RX2, RS1, RS2, SI3, SI4, SS1, SS2

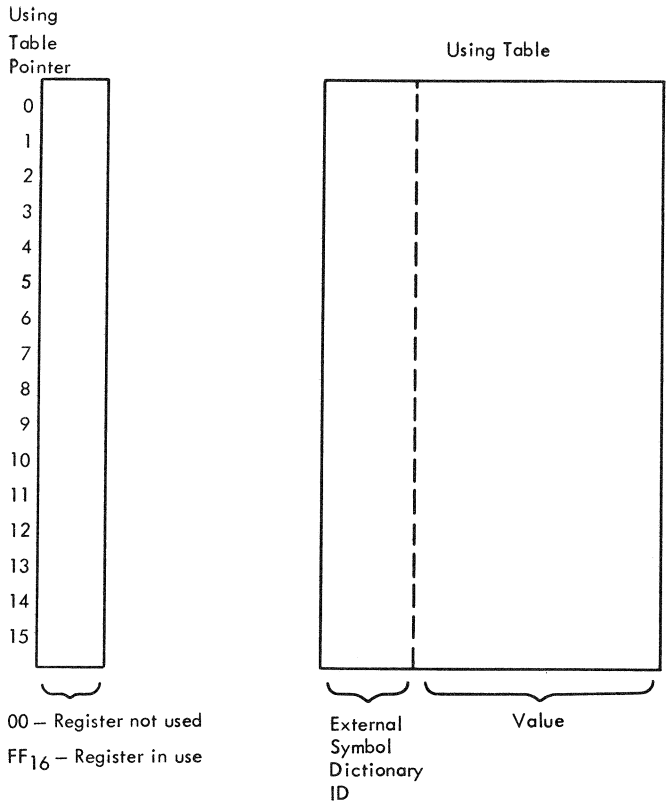
These subroutines scan the operand fields of machine operations. They do a syntax check and format the instruction building area, F8INST(ACT). See Figure 15. These subroutines call on lower level subroutines which do semantic checks, and call on another set of subroutines which do expression evaluation and decomposition.

RR4 refers to extended mnemonic register-to-register instructions. RX2 refers to extended mnemonic register-to-indexed-storage instructions.

F8AREX does a complete syntactic scan on each expression and sets a group of semantic flags. The necessary flags are investigated by the individual routines, and error procedures are taken where necessary.

IEUF8A - ASSEMBLER OPERATION PROCESSOR (FLOWCHART 31)

IEUF8A processes the following assembler ops:



The using table is used by the decomposition routine. The using table pointer and the using table are parallel tables. Every decomposable value is checked against the complete table for possible decomposition.

Figure 14. Decomposition Routine Using Table

MNOTE, PRINT, SPACE, EJECT, PUNCH, REPRO, TITLE, ENTRY, EXTRN, WXTRN, START, CSECT, DSECT, COM, EQU, ORG, END, LTORG, USING, DROP, literal DC, DC, DS, CCW, and CNOP.

IEUF8A is entered from the IEUF8C routine. FRB (register 12) is loaded with the base address of IEUF8A, and control is passed to the assembler control table where registers 4, 5, 6, and 7 are saved. Control is then passed to the address in FRB.

IEUF8A SUBROUTINES

PRINTB

This routine investigates the operand field of all print statements. A syntax check is made, and from one to three conditions are set per statement.

ON	sets	F8PON(ACT)	to	0
OFF	sets	F8PON(ACT)	to	FF_{16}
GEN	sets	F8PGEN(ACT)	to	0
NOGEN	sets	F8PGEN(ACT)	to	FF_{16}
DATA	sets	F8PDAT(ACT)	to	0
NODATA	sets	F8PDAT(ACT)	to	FF_{16}

SPACE

The SPACE routine does a complete syntax check and, if necessary, checks semantics, and does expression evaluation. A switch, SPACSW(ACT), is set to FF_{16} on a valid SPACE statement and to AA_{16} on an error condition. SP2 (register 11) is set to the number of lines to be spaced.

EJECT

In this routine, a switch, EJCSTW(ACT), is set to FF_{16}.

PUNCHB

In this routine, a switch, REPSW(ACT), is set to 1.

REPRO

In this routine, a switch, REPSW(ACT), is set to 3.

TITLEB

In this routine, a switch, REPSW(ACT), is set to 7.

MNOTST

In this routine, a switch, REPSW(ACT), is set to 15.

ENTRYB

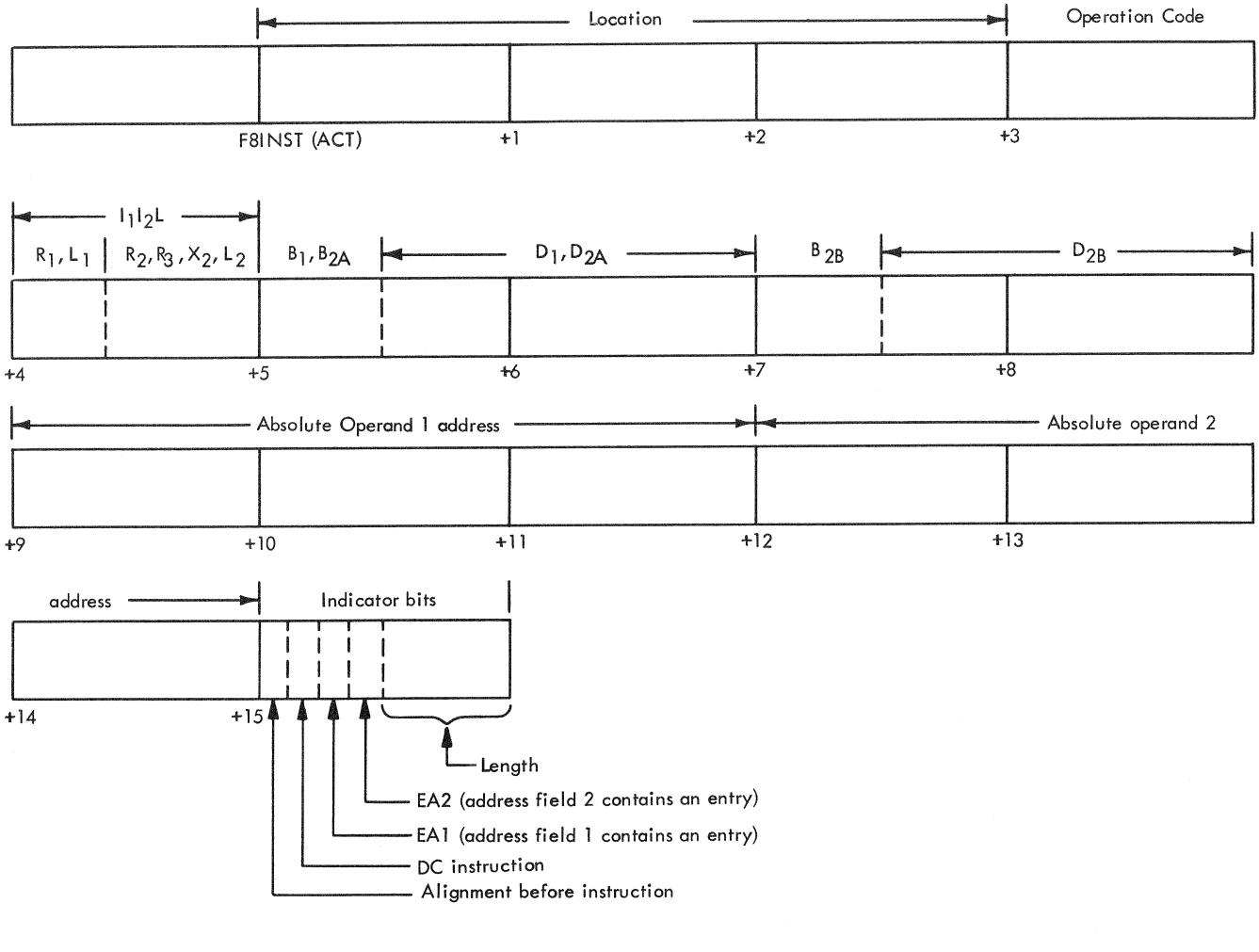
The ENTRYB routine in Phase F8 is only an error checking routine. All actual processing has been completed in Phase F7.

EXTRNB

The EXTRNB routine returns control to mainline control. EXTRN and WXTRN processing is completed in Phase FI.

STARTB

The start routine checks the private code switch, CTPCSW(ACT). If private code is not initiated, the non-reentrant switch is turned on. If private code has been initiated, the start statement is ignored, and control is immediately returned to Phase F8 main line control.



●Figure 15. Instruction Building Area

CSECTB

The CSECTB routine checks if the new external symbol dictionary ID is different from the current external symbol dictionary ID. If not, control is immediately returned to Phase F8 mainline control. If different, current type, external symbol dictionary ID, and current adjustment base are set, and the non-reentrant switch is turned on.

DSECTB

The DSECTB routine checks if the new external symbol dictionary ID is different from the current external symbol dictionary ID. If not, control is immediately returned to Phase F8 mainline control. If different, current type and external symbol dictionary ID are set. Current adjustment base is set to zero, and the non-reentrant switch is turned off.

COMB

The common routine is the same as the DSECTB routine. The non-reentrant switch is turned on.

NOTE: The reentrant error switch, CTRENT-(ACT), is a one-byte cell set by an initialization routine to zero. A violation of reentrant code is detected by IEUF8M, and the associated error routine sets CTRENT(ACT) to FF_{16}. The switch is investigated by Phase FPP, and, if the setting FF_{16} has occurred, the appropriate error message is logged. No message for possible reentrant error occurs in line.

CCWB

The CCWB routine processes the operand field of CCW statements. A complete syntax and semantic check is made. The instruction building area is formatted for output.

Through the IEUF8D routine, relocation dictionary entries are made for the second operand of CCW statement.

CNOPB

This routine places as many BCR instructions as necessary in the instruction stream.

EQUB

The equivalance routine puts the equated value into the address location for printed output.

ORGB

The ORGB routine returns control to mainline control. Processing is done in Phase F7.

ENDB

The ENDB routine does a syntax check and sets the end switch ENDSWH(ACT) for the Post Processor to put out the correct end card. It also formats the printout of the end record.

LTORGB

This routine tests to see if literals are to be put out. If they are, the literal adjustment table is input, and the location field is set to the address of the first literal to be output.

USINGB

This routine does a complete syntax and semantic check on the operand of a using statement. A using table pointer is set to FF_{16} for each register being used, and the correct value and the external symbol dictionary ID is set in the corresponding Using table entry.

DROPB

This routine does a complete syntax and semantic scan on the operand field of a DROP statement. The Using table pointer is set to 0 for each register dropped.

LITERB, DSB, DCB, DXD, CXD

Switches are set for proper branching within the IEUF8D routine, and control is passed to IEUF8D.

IEUF8P - OUTPUT ROUTINE (FLOWCHARTS 32-33)

At entry, the current text record at F8WORK(ACT) is moved into INPUT work area. If this record is one of the edited types, a partially formatted left side at F8INST (ACT) is moved into LFTHLF work area.

IEUF8P has four entry points: IEUF8P, BLDIMG, COMMENT, and LOADRA. Text records are passed to IEUF8P as they are encountered in the text stream.

- IEUF8P processes edited text records (type 100).
- BLDIMG builds a source image from generated edited records (types 110 or 111).
- COMMENT loads the right side for source records (type 000, 010, or 011), puts an error line for error records (type 001), and saves source records (if TERM option in effect).
- LOADRA prints an error line if the last record is an error record (type 001).

As an example, if the PRINT ON, DATA, and GEN options have been requested, the following action is taken (as applicable):

- IEUF8P entry (type 100 records) - The left side of the print line buffer is loaded, and the entire buffer is dumped.
- BLDIMG entry (type 110 or 111 records) - A source image is constructed and treated like a source record (type 010 or 011), which is then processed like an edited record (type 100).
- COMMENT entry (type 000, 010, or 011 records) - The previous right side of the print line buffer is printed (if still loaded), and the information is put in the punch buffer. The right side of the print buffer is loaded with the current record.
- COMMENT entry (type 001 record) - An error line is printed if the error indicator is on.
- LOADRA entry (type 001 record) - The error record is reformatted to include the statement number, and an error indicator is turned on.

IEUF8P has the following functions (acronyms within parantheses refer to flowchart terms):

- Format left side (LOADLH) and right side (LOADRH) of print line and print entire line (CHKSWH).
- Format text output cards (GOTXT) and punch them (DUMP).

- Build source image from generated edited records (BLDIMG).
- Put title in page heading and put page heading for each new page (PGEHED).
- Reformat error record to include the statement number (WTERR) and print error line (LOADRERR).
- Space when SPACE is encountered and eject to a new page for EJECT and TITLE.
- Format and print MNOTE message.
- Punch REPRO and PUNCH cards (SYSO).

IEUF8P formats are given in Figure 16.

Print Heading Buffer (Figure 16)

H1 DECKNM - Deck identification.

HEADNG - Programmer heading from TITLE statement.

PAGE - "PAGE".

PCNT - Page number.

H2 Col. headings for printed listing.

Print Line Buffer (Figure 16)

LOC - Address of instruction relative to start of program.

OP - Hex op code.

R1R2 - Registers specification.

OPN1 - 2nd instruction byte.

OPN2 - 3rd instruction byte.

EA1 - Relative address of operand 1.

EA2 - Relative address of operand 2.

DSTMNT - Statement number.

SOURCE - Source statement.

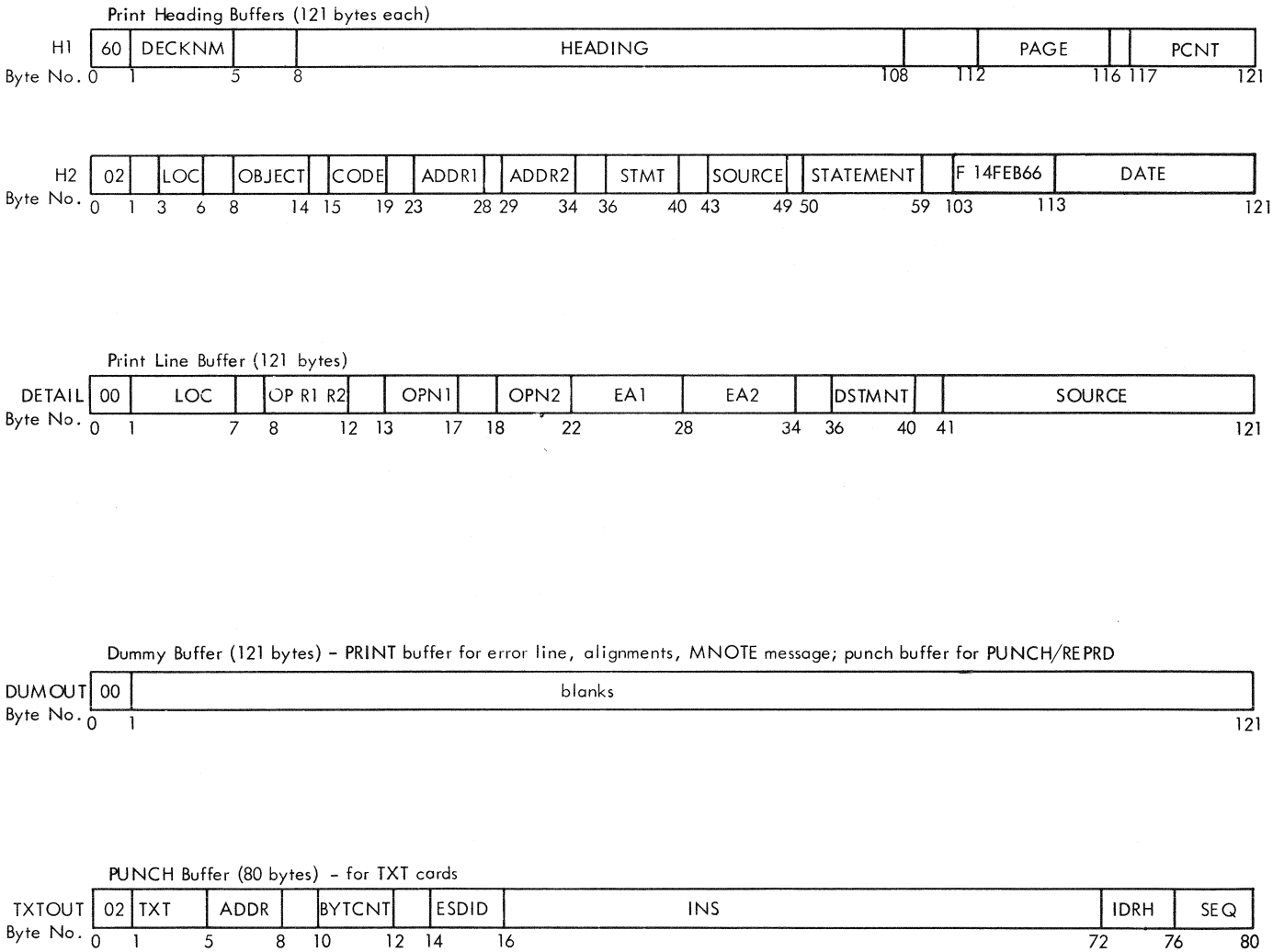
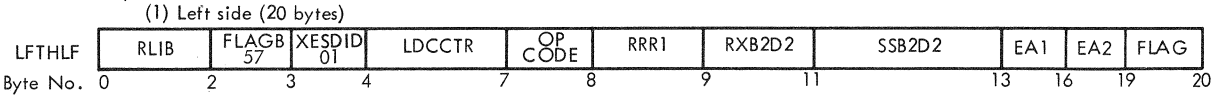


Figure 16. IEUF8P Formats

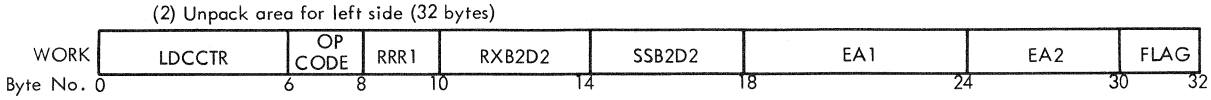
Input Buffers

(1) Left side (20 bytes)

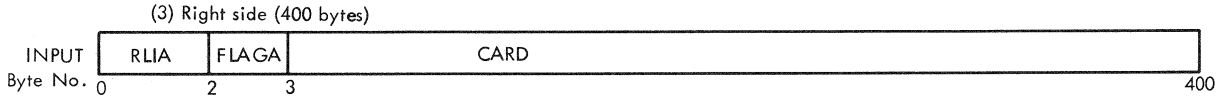


Indicator	Settings
1111	DS statement
1000	Align. data
0100	DC statement
0010	EA1 present
0001	EA2 present
00XX	Syllable count
00	1 - 2 bytes
01	2 - 4 bytes
10	2 - 4 bytes
11	3 - 6 bytes

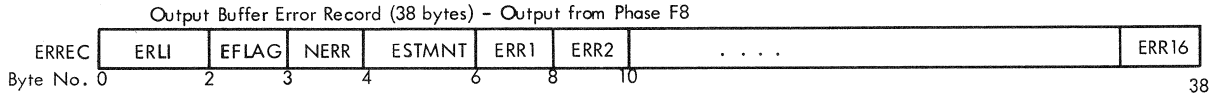
(2) Unpack area for left side (32 bytes)



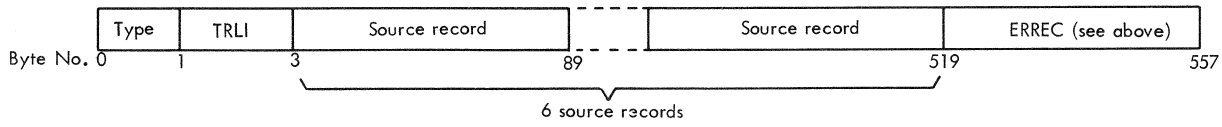
(3) Right side (400 bytes)



Output Buffer Error Record (38 bytes) - Output from Phase F8



SYSTERM Buffer - optional



• Figure 16. IEUF8P Formats (Cont'd)

Punch Buffer for TXT Cards (Figure 16)

TXT - "TXT"

ADDR - Relative address of 1st instruction on card.

BYTCNT - Number of bytes of information on card.

ESDIDX - External symbol dictionary number.

INS - 56 bytes of information.

IDRH - Deck identification.

SEQ - Card sequence number.

Input Buffer/Output Buffer (Figure 16)

RLIB (RLIA/ERLI/TRLI) - Record length indicator.

FLAGB (FLAGA/EFLAG) - Record type.

XESDID - External symbol dictionary number.

LOCCTR - Relative address of instruction.

OPCODE - Hex op code.

RRR1 - Registers specification.

RXB2D2 - 2nd instruction byte.

SSB2D2 - 3rd instruction byte.

EA1 - Relative addresses of operand.

EA2 - Relative addresses of operand.

FLAG -

CARD - Source statement.

NERR - Number of errors in statement.

ESTMNT - Statement number.

ERR1 - ERR16 - 1-16 error codes and pointers.

IEUF8D - DC EVALUATION (FLOWCHARTS 34-35)

Phase F8 mainline control calls IEUF8D each time a DC, DS, literal DC, or CXD is encountered. The low order byte of register 13, at entry, is set as follows: 00 = DS, FF_{16} = DC, mixed = literal DC.

IEUF8D processes an entire DC statement each pass. The print routine, IEUF8P, is called as many times as necessary. There is one DC work bucket (15 bytes) in the appended fixed field for each operand in a statement, and each operand may contain one or more constants. See Figure 11. Exceptions are character, hexadecimal, and binary type DCs which may contain only one constant per operand.

Relocation dictionary entries are made for A-, Y-, V-, and Q-type relocatable constants which meet minimum length specifications. However, no relocation dictionary entries are made for address constants within a dummy section or common, nor for address constants whose operand address is within a dummy section. One relocation dictionary entry is made for each CXD instruction providing that the CXD is not in a dummy section or common.

Upon completion of a statement, IEUF8D returns to IEUF8C by executing a branch to location CTRTRN in the central control table.

IEUF8N - PHASE F8 FLOATING AND FIXED-POINT CONVERSION (FLOWCHART 36)

This routine does all floating - and fixed-point conversion in declarative (DC) statements. It is called by the Phase F8 DC evaluation routine (IEUF8D) once for each constant in a floating-point or fixed-point DC operand. At entry, Register No. 1 points to the first byte of the constant to be converted (the first byte past the left delimiter), and Register No. 10 points to the corresponding DC work bucket in the edited text record. The DC work bucket contains the DC type, length modifier, scale factor, and external exponent modifier.

After the conversion has been completed, program control is returned to IEUF8D evaluation with three pointers: Register No. 1 points to the right delimiter, Register No. 2 points to the converted value of the constant (16 bytes), and Register No. 13 points to an error flag. If Register No. 13 is zero, no error has occurred.

IEUF8V - EXPRESSION EVALUATION SUBROUTINE

This subroutine is the same as IEUF7V with the exception of XREF, but the two are loaded separately in their respective phases.

IEUF8L - LOG ERROR SUBROUTINE

This subroutine is the same as IEUF7L, but the two are loaded separately in their respective phases.

IEUF8S - SYMBOL TABLE SUBROUTINE

This subroutine has one entry point, STGETR. This tests whether the requested symbol is in the table and, if so, gives the address of the first byte in the entry after the name field. If not, zero is returned.

OVERALL OPERATION (FLOWCHARTS 37 AND 38)

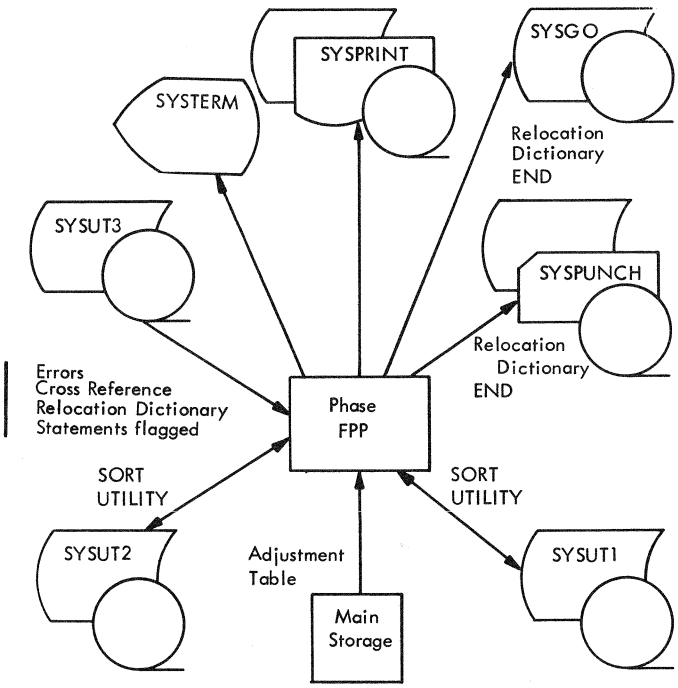
The Post Processor Phase, Phase FPP, is divided into two control sections which are executed serially.

The first section, IEUFPP, reads the RLD and cross-reference records from the overflow file (SYSUT3), sorts them, writes RLD on SYSPUNCH and/or SYSGO and SYSPRINT and the XREF records on SYSPRNT. (If the DOS assembler option is specified, the RLD records will not be sorted. See Appendix A.) FPP also writes the END object card on SYSPUNCH and/or SYSGO after the RLD. See Figure 17. In the event there are too many relocation dictionary/cross-reference record entries to be held in main storage at one time, SYSUT1 and SYSUT2 are used for sorting.

The second section, IEUFD, reads error records* from the overflow file (SYSUT3) and formats an error message by matching a code in the error record against a message table in main storage. IEUFD also prints the statistical information (statements flagged, highest severity code, and line count) and returns to the invoker through RTA and IEUASM.

IEUFPP FUNCTIONS (FLOWCHART 37)

The IEUFPP functions are designed to produce the relocation dictionary and the cross-reference list, if requested.



● Figure 17. I/O Flow for Phase FPP

* together with the flagged statements if the TERM option is in effect. It then also produces the diagnostic information on SYSTERM.

IEUFPP sorts the relocation dictionary entries by address. The sorted dictionary is written onto SYSPRINT and SYSPUNCH and/or SYSGO. The loader END record is constructed and written onto SYSPUNCH.

If the EXEC statement did not specify NOXREF, IEUFPP sorts the cross-reference table entries by symbol. The sorted table is written on SYSPRINT.

IEUFPP transfers control to Section IEUFD by branching.

IEUFPP SUBROUTINES

CHKSWH - Jump Table Sort

FPP - Phase Initialization (IEUFPP)

EPRLZ - Merge Tape Writer

EP2 - Merge Tape Writer

ESORT - Input Data Sorter

GTOR - Control Table Reader

GTOX - Cross Reference List Checker

This routine checks for the cross-reference list output option.

PPIN - Phase PP Initialization

RD1RLD - Relocation Dictionary Record Merger

RD1XRF - Cross-Reference Record Merger

READR - Relocation Dictionary Reader

This routine reads the relocation dictionary records from input tape and stores them.

READX - Cross-Reference Reader

This routine reads cross-reference records from input tape and stores them.

SETOT1 - Relocation Dictionary Writer

This routine writes all relocation dictionary data from main storage on SYSPRINT, SYSPUNCH, and/or SYSGO.

SETOT2 - Cross Reference Writer

This routine writes all cross-reference data from main storage.

WR1RLD - Relocation Dictionary Writer

This routine writes the relocation dictionary string on SYSUT1 (or SYSUT2) for sorting.

WR1XRF - Cross-Reference Writer

This routine writes cross-reference on SYSUT1 (or SYSUT2) for sorting.

XRFLOD - Cross-Reference Starter

This routine starts the cross-reference input pass.

IEUFD FUNCTIONS (FLOWCHARTS 38)

The Phase FPP diagnostic, IEUFD, 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 and SYSTERM.

The functions of this phase are as follows:

- Write source records (on SYSTERM).
- Write diagnostic messages (on SYSTERM and SYSPRINT).
- Accumulate and print the total number of error messages in the entire assembly (on SYSTERM and SYSPRINT).
- Process and print the highest severity code (on SYSTERM and SYSPRINT).
- Print statistics and options in effect (on SYSPRINT).
- Print the line count (on SYSPRINT).

If the TERM option is in effect, the SYSTERM data set is opened (this data set is used at the PUT level). 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 65,536 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, the number of printed lines in the listing is printed, and the phase exits to IEURTA. 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 and printed. The phase exits to IEURTA after printing the line count.

IEUFD SUBROUTINES

FD

FD 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 Y-TYPE CONSTANT. . ." message. If there are no relocatable Y-type constants in the program, it branches to ML00.

RDERR - Error Record Reader

HCC - Severity Code Storage

This routine stores, in register 15, the highest severity code encountered.

ML00 - Error Record Tester

This routine 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 ML01A.

ML01 - Error Record Getter

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

ML01A - Error Statement Getter

This routine gets the error statement number and accumulates an error statement total.

ML01B - Error Statement Converter

This routine converts the error statement number into decimal for listing and points to the appropriate error message.

ML03 - Error Message Converter

This routine converts the error message for listing and lists it.

ML05 - Error Statement Comparer

This routine compares the error statement severity code to the highest severity code yet encountered. It saves the new severity code if it is higher than any previously encountered.

ML10 - Statement Printer

This routine prints the total number of statements flagged.

ML11 - Error Message Lister

TRMXRTN - SYSTERM DCB Exit Routine

TREDIT - SYSTERM Source Statement Editor

If the TERM option is in effect this routine arranges SYSTERM output depending on STMT/NOSTMT and NUM/NONUM options.

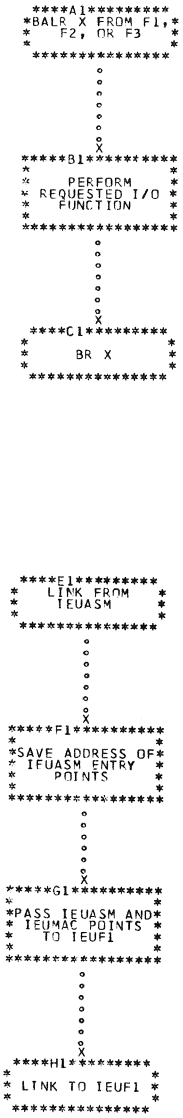
SYSTRMD - SYSTERM Output Lister

PHASE ERR - PERMANENT I/O ERROR ABORT

The I/O error abort phase, Phase ERR, obtains information about a permanent I/O error and then terminates the assembly. It consists of one control section -- IEUERR -- which receives control (via XCTL) from the phase in which the error occurs. That phase frees the core it originally obtained and waits for completion of other outstanding I/O requests before passing control to IEUERR.

IEUERR issues a SYNADAF macro instruction to get the following detailed information about the error -- jobname, stepname, unit address, device type, ddname, operation attempted, and error description. This information is output to either the console device or SYSPRINT.

IEUERR returns to either MAC or RTA with a Return Code of 20. Figure 3 shows the control flow to and from Phase ERR.



• Chart 1. IEUMAC - Macro Generator I/O

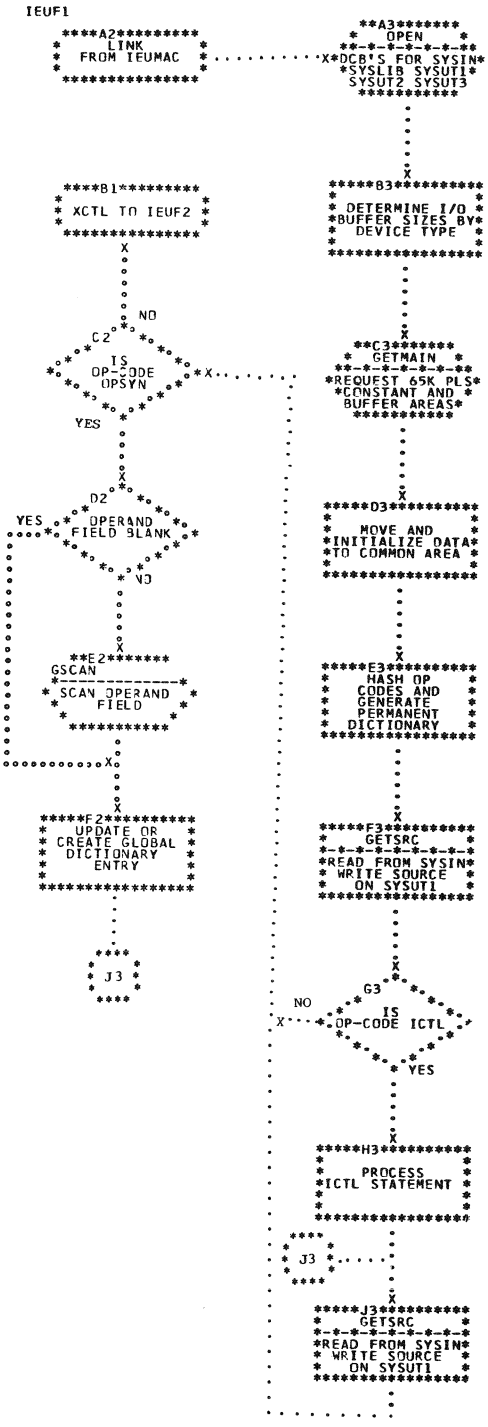


Chart 2. IEUF1 - Phase F1

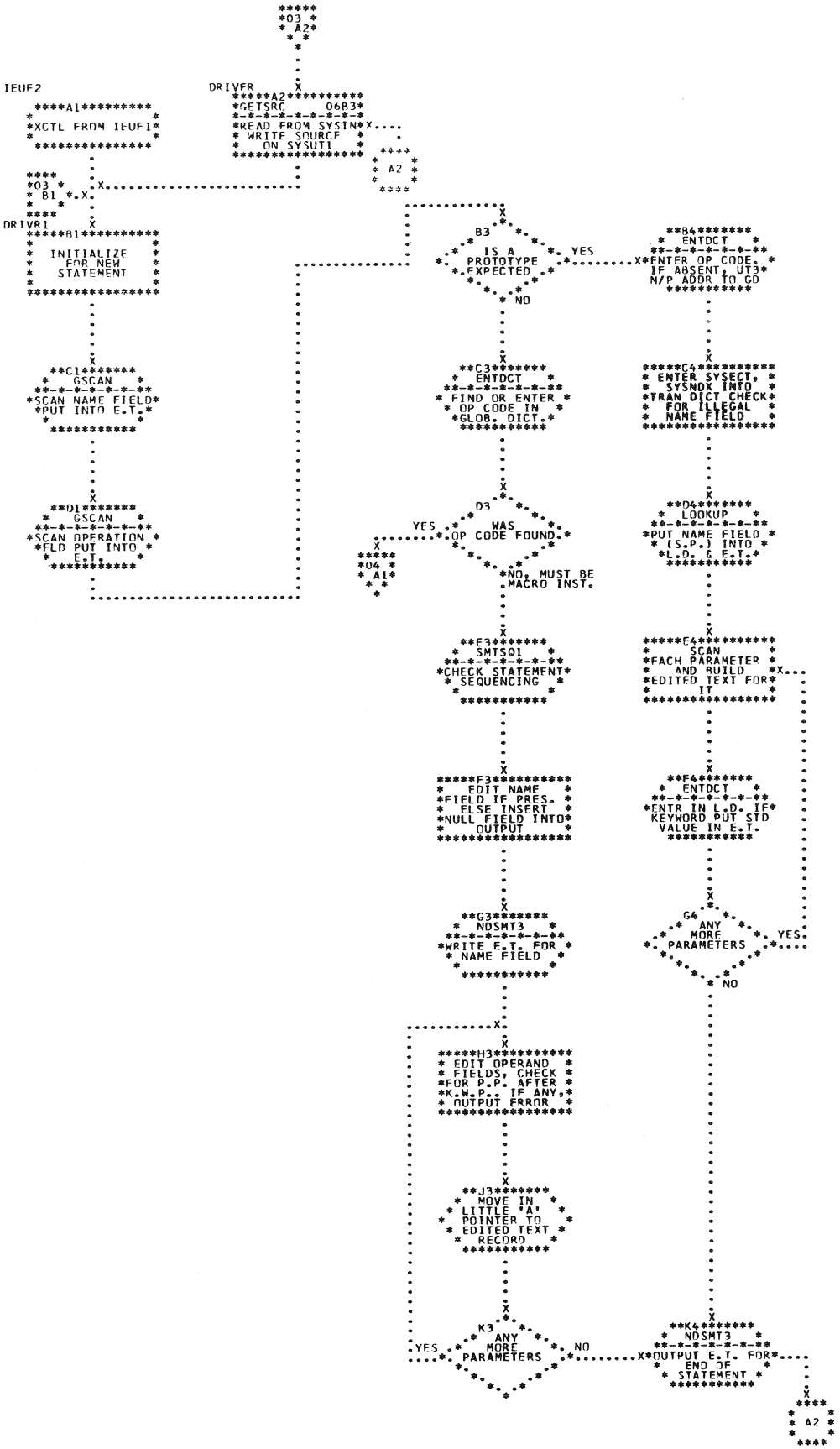


Chart 3. IEUF2 - Phase F2 (1 of 5)

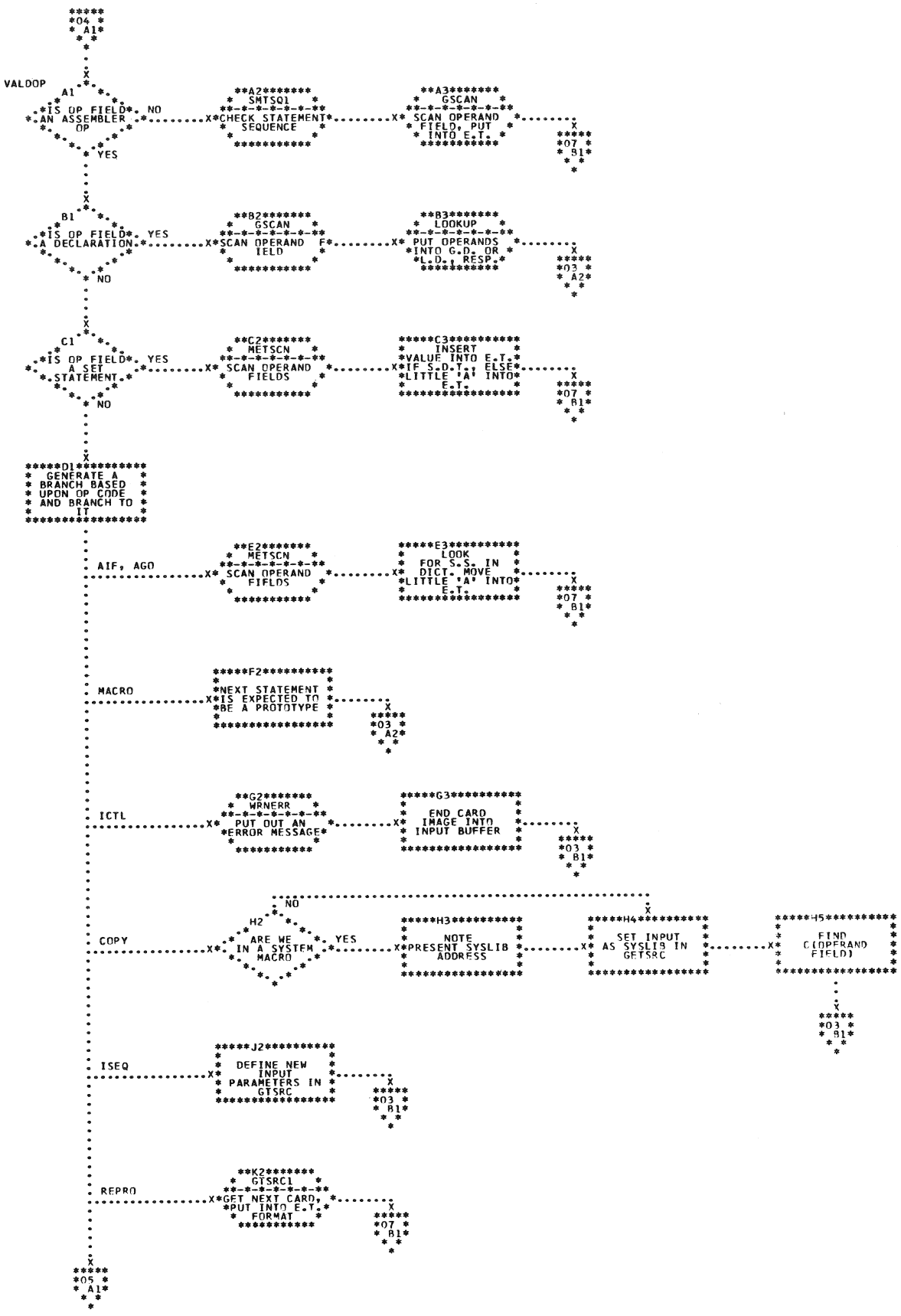
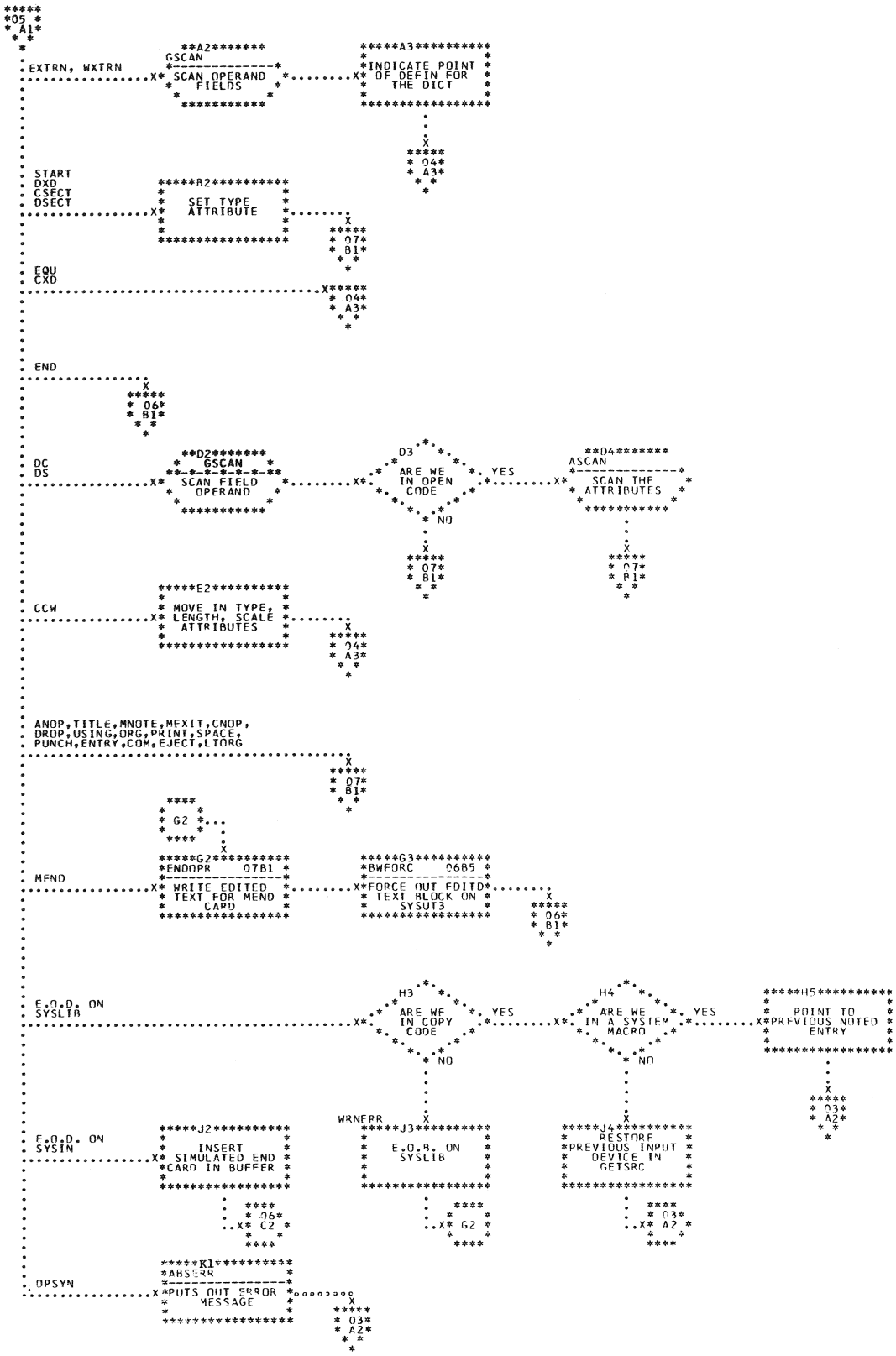


Chart 4. IEUF2 - Phase F2 (2 of 5)



●Chart 5. IEUF2 - Phase F2 (3 of 5)

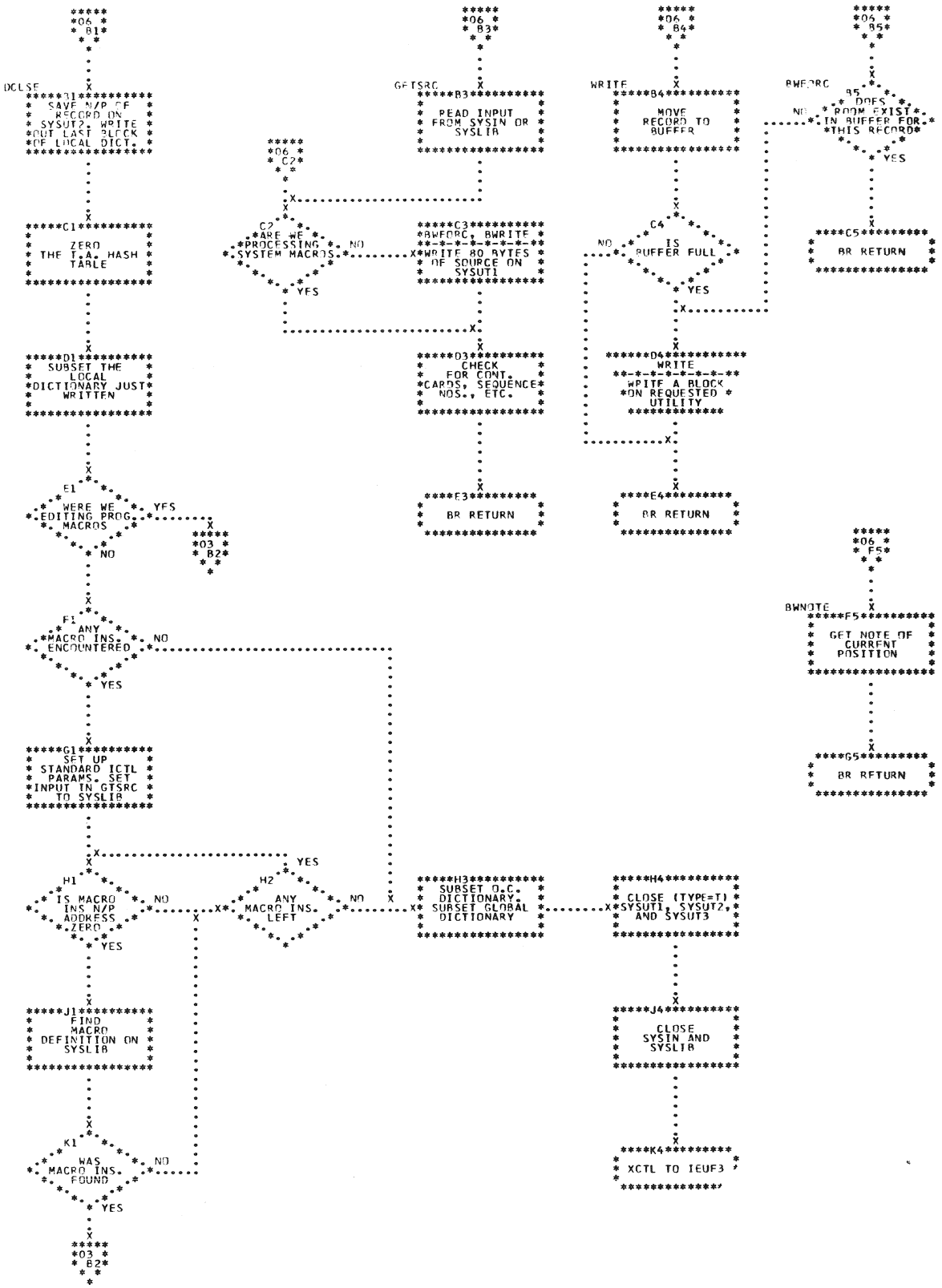


Chart 6. IEUF2 - Phase F2 (4 of 5)

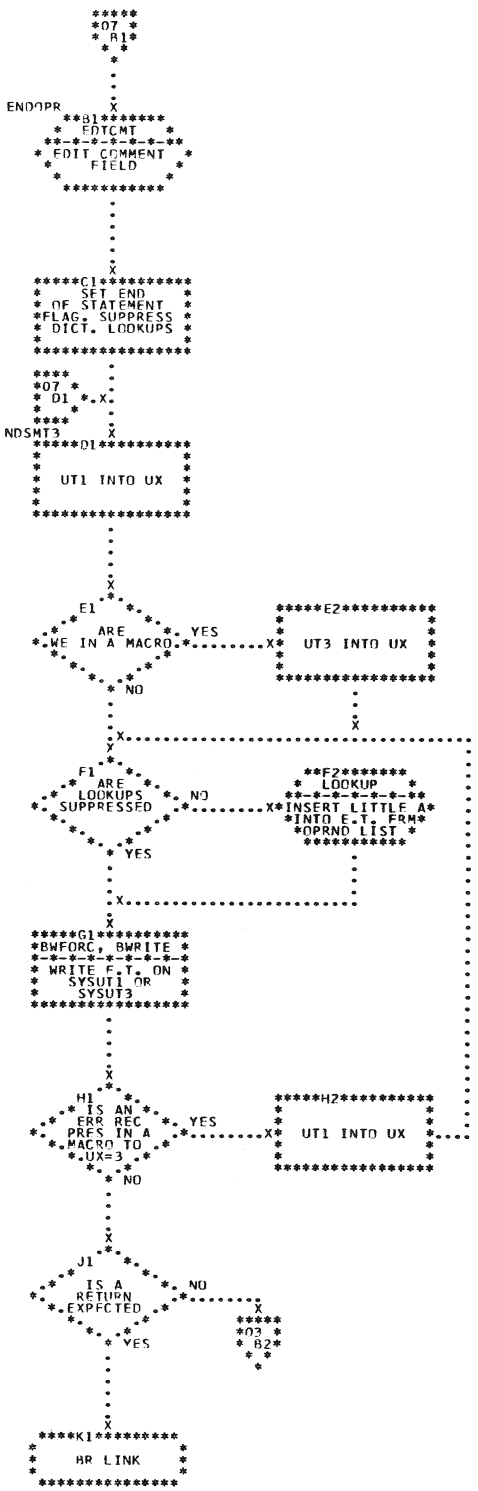


Chart 7. IEUF2 - Phase F2 (5 of 5)

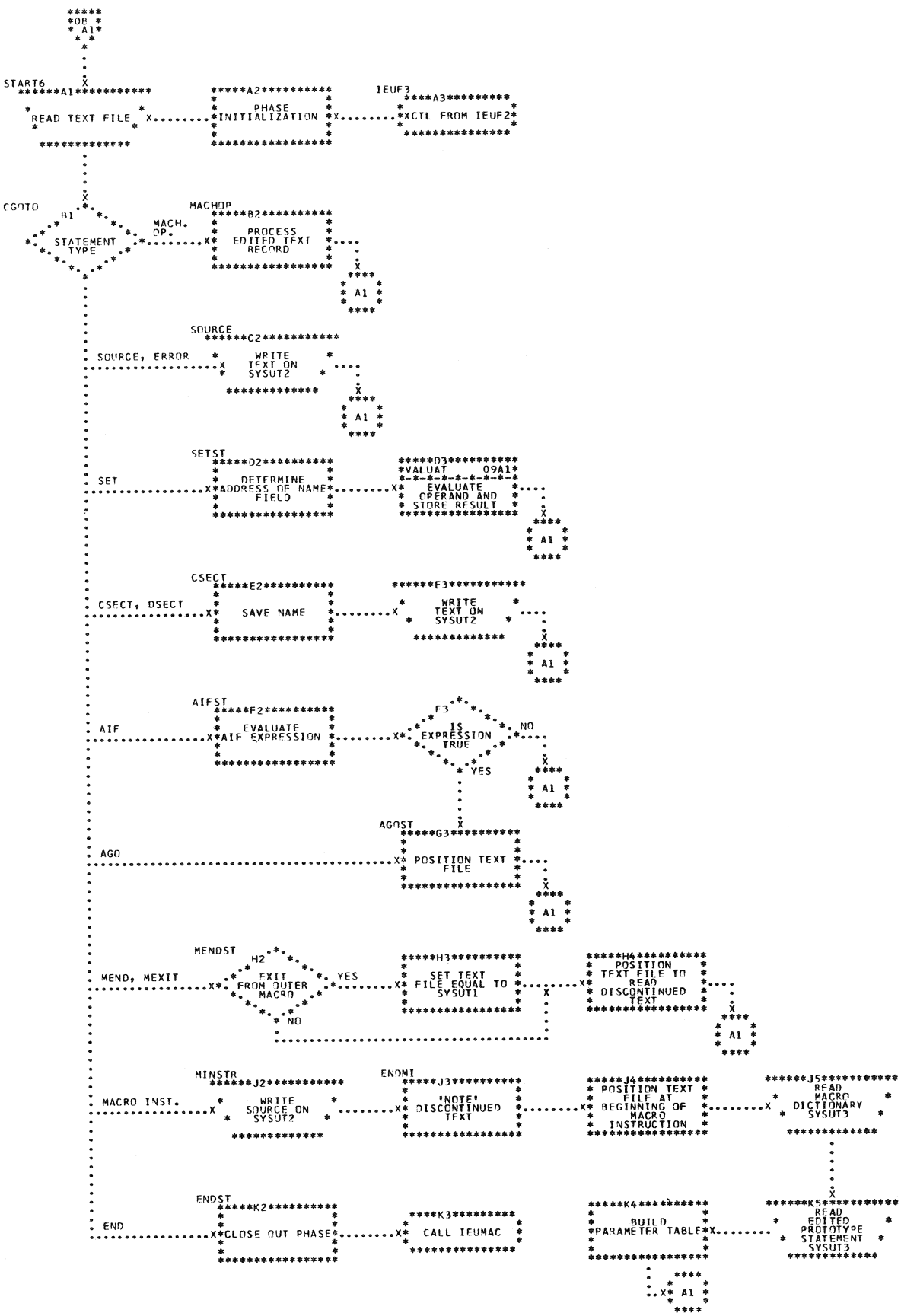


Chart 8. IEUF3 - Phase F3 Main Line Control

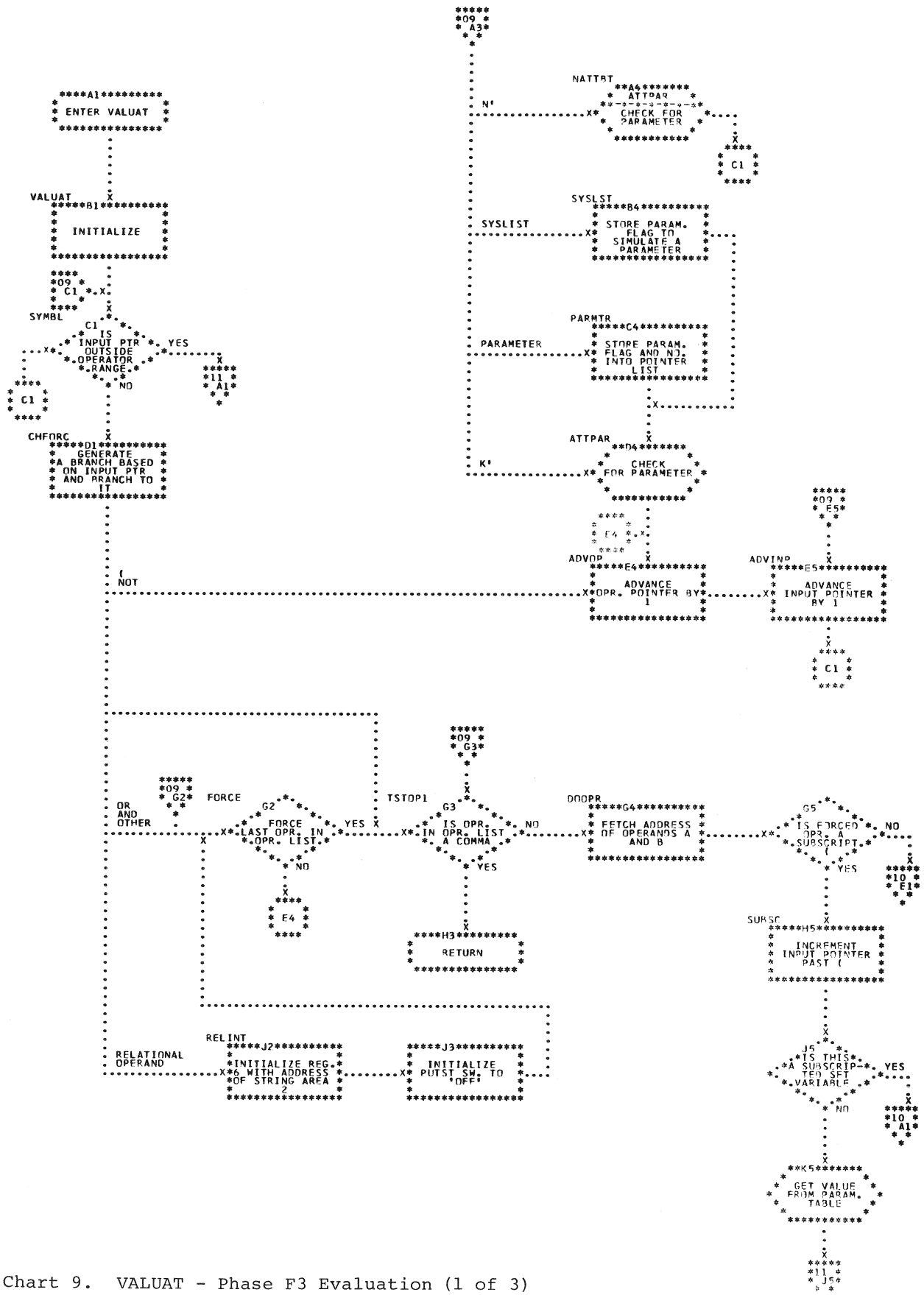


Chart 9. VALUAT - Phase F3 Evaluation (1 of 3)

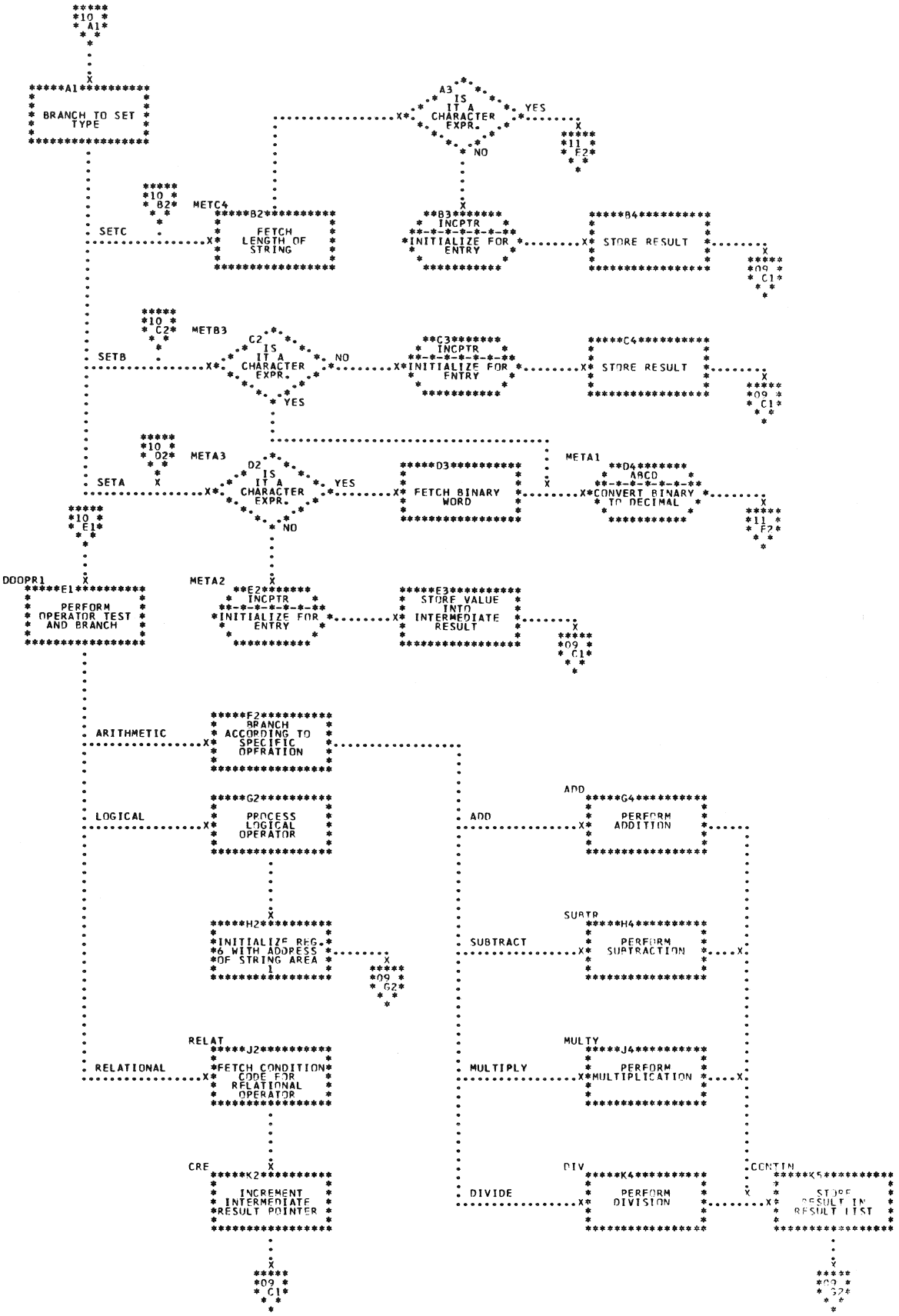


Chart 10. VALUAT - Phase F3 Evaluation (2 of 3)

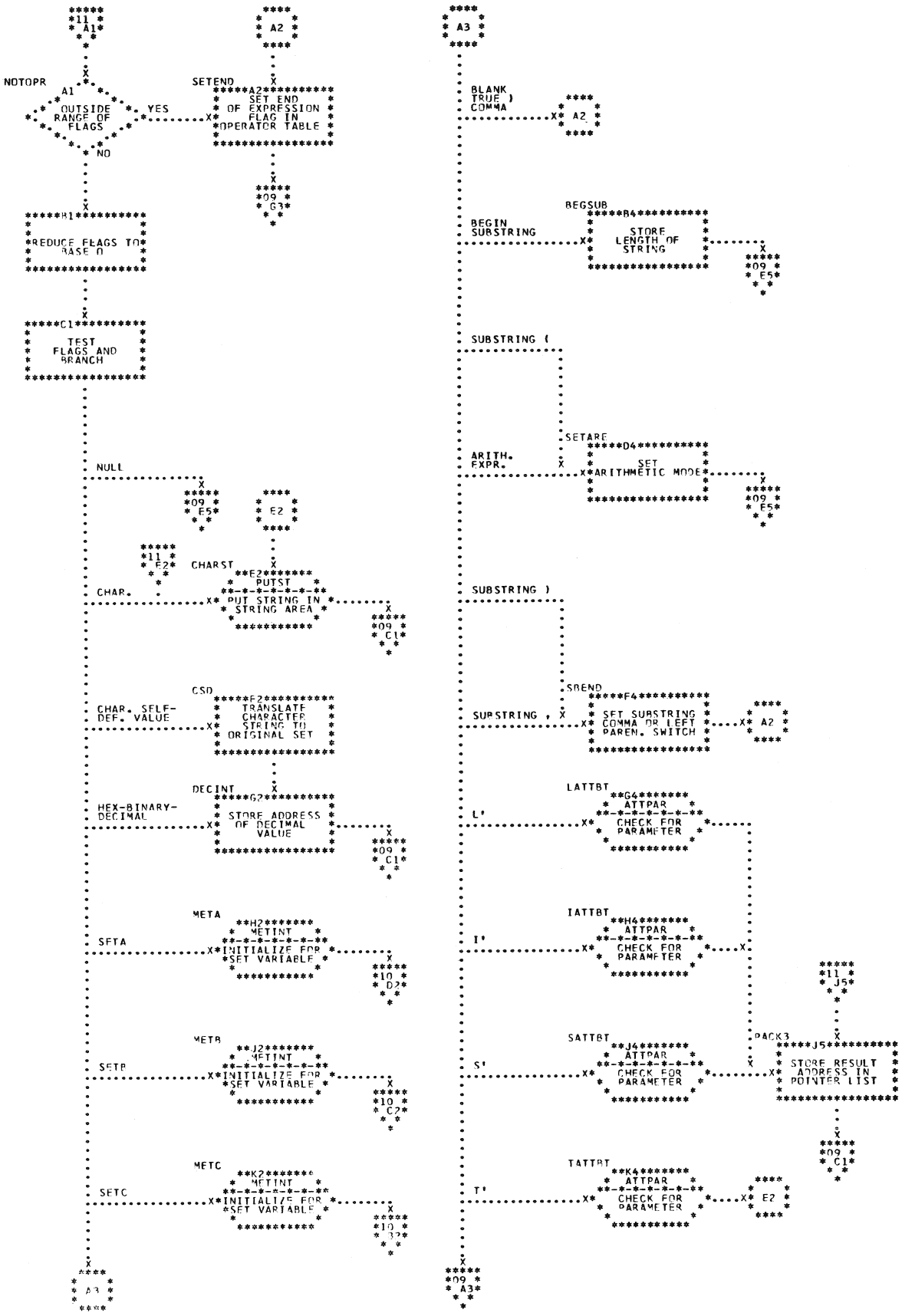


Chart 11. VALUAT - Phase F3 Evaluation (3 of 3)

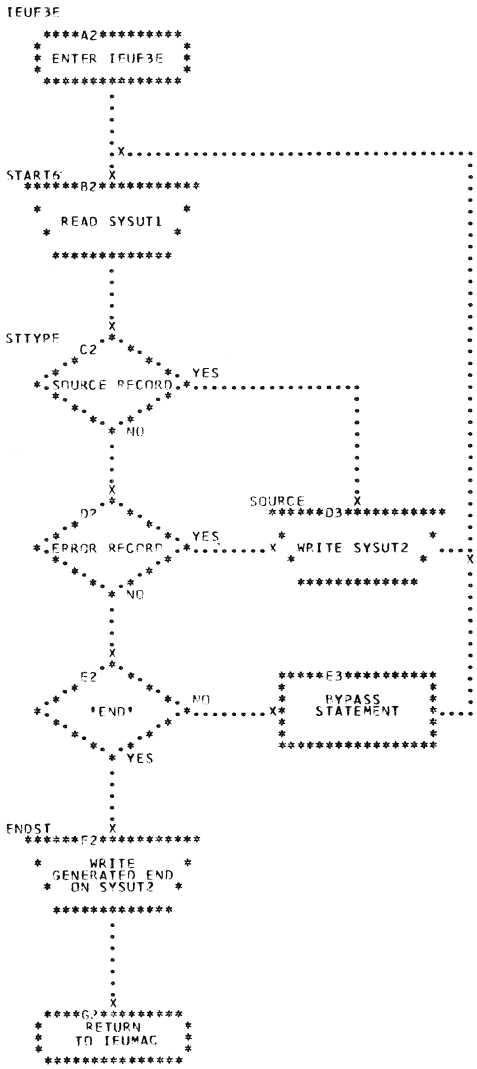


Chart 12. IEUF3E - Phase F3 Substitute

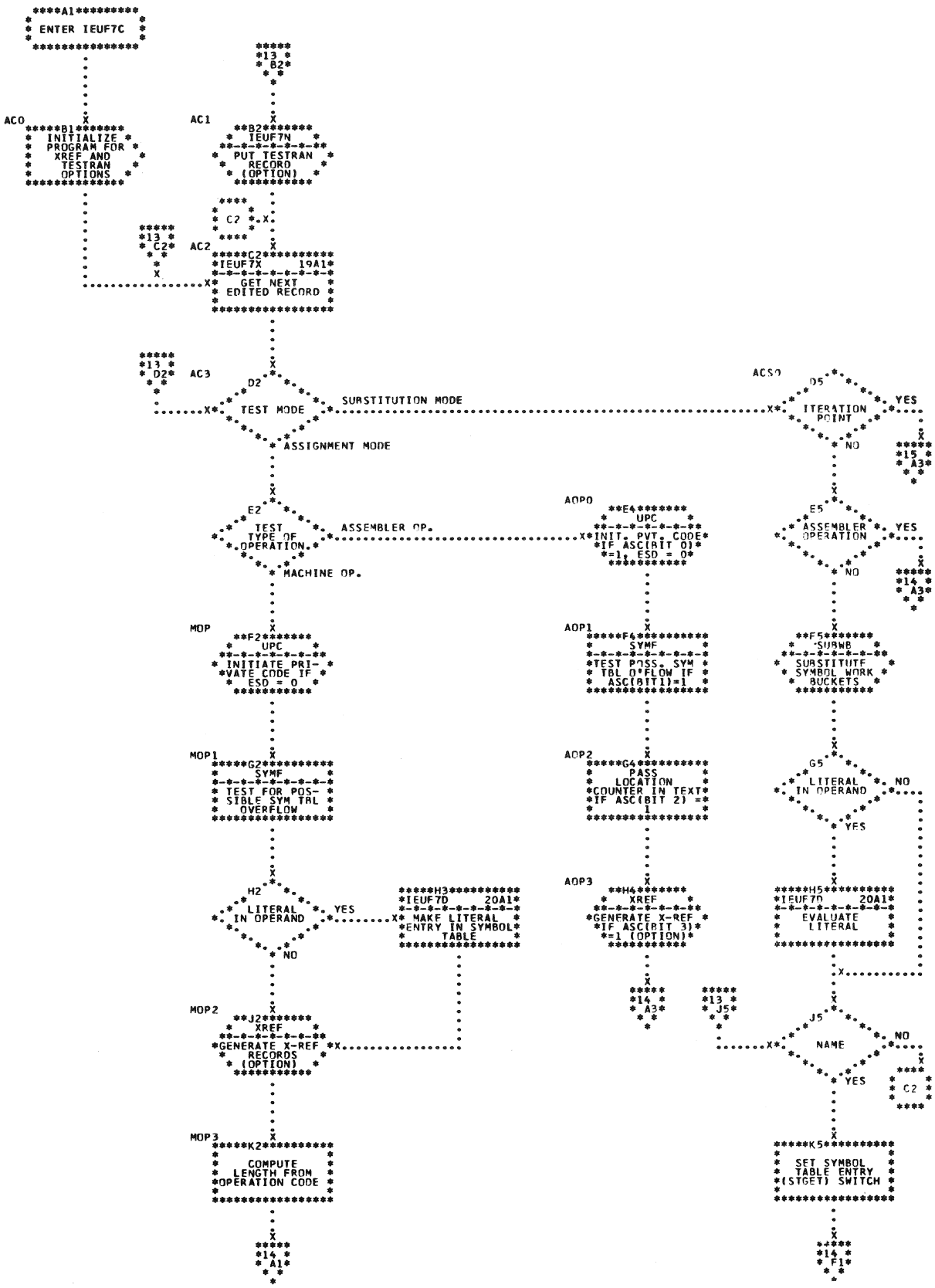
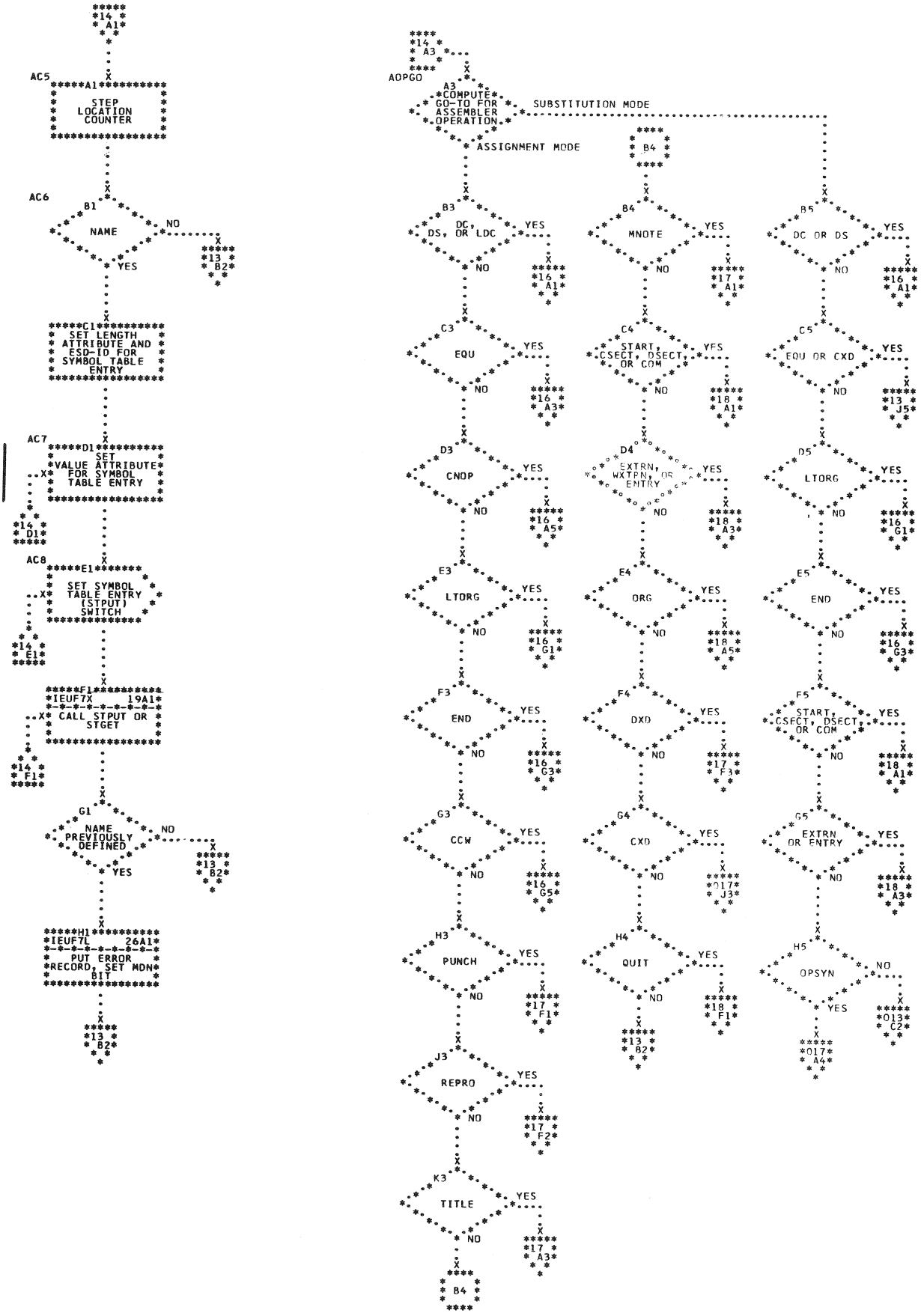


Chart 13. IEUF7C - Phase F7 Main Line Control (1 of 6)



●Chart 14. IEUF7C - Phase F7 Main Line Control (2 of 6)

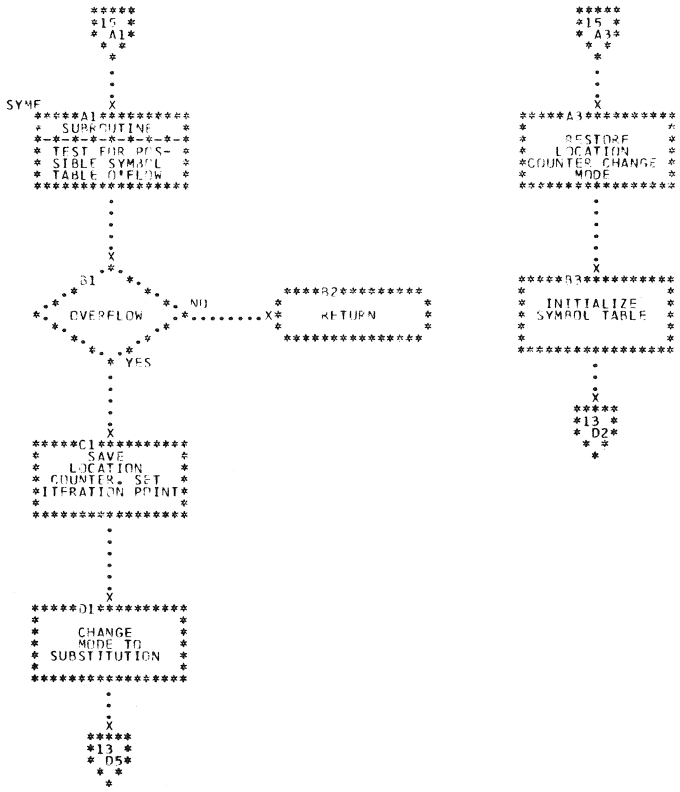


Chart 15. IEUF7C - Phase F7 Main Line Control (3 of 6)

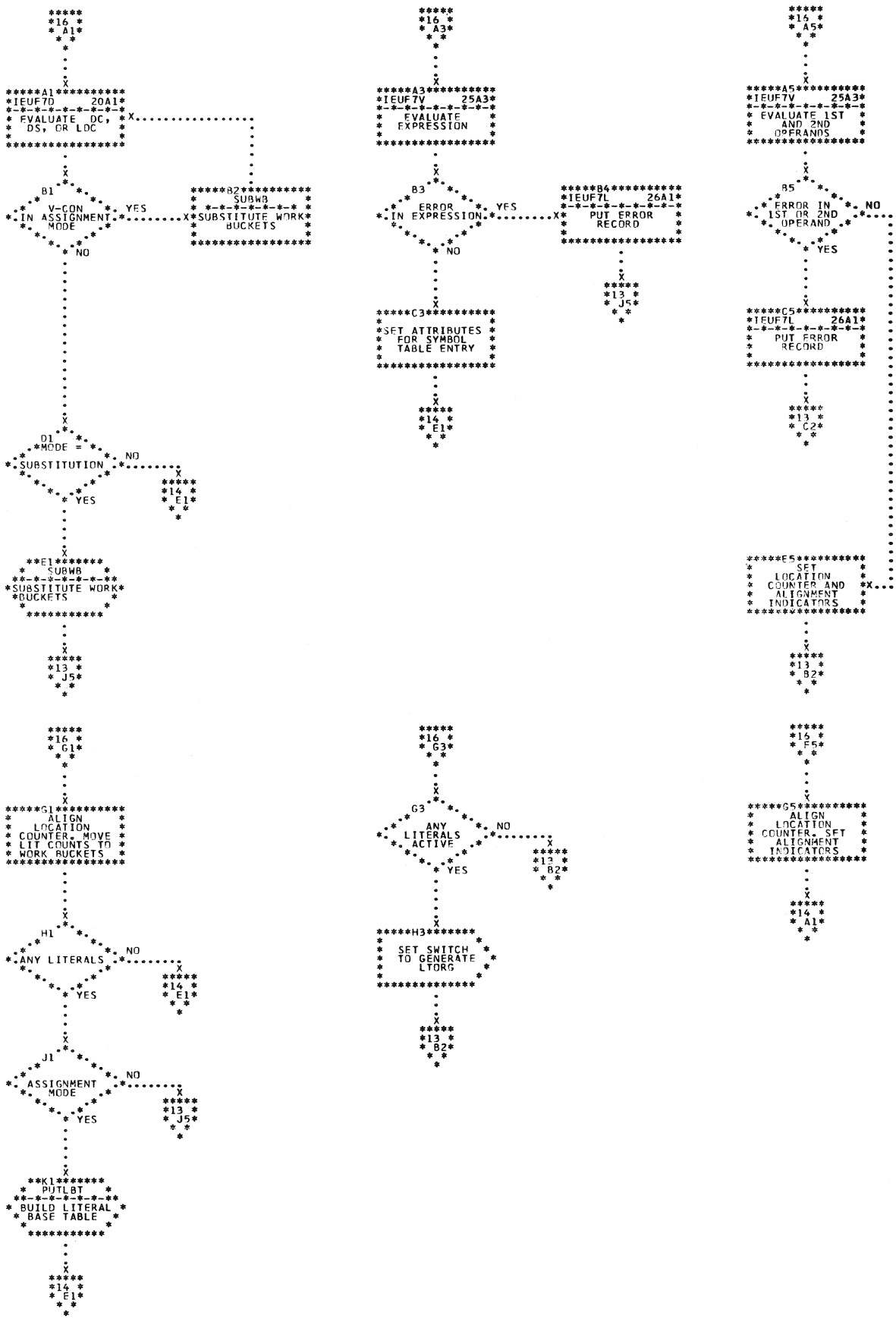


Chart 16. IEUF7C - Phase F7 Main Line Control (4 of 6)

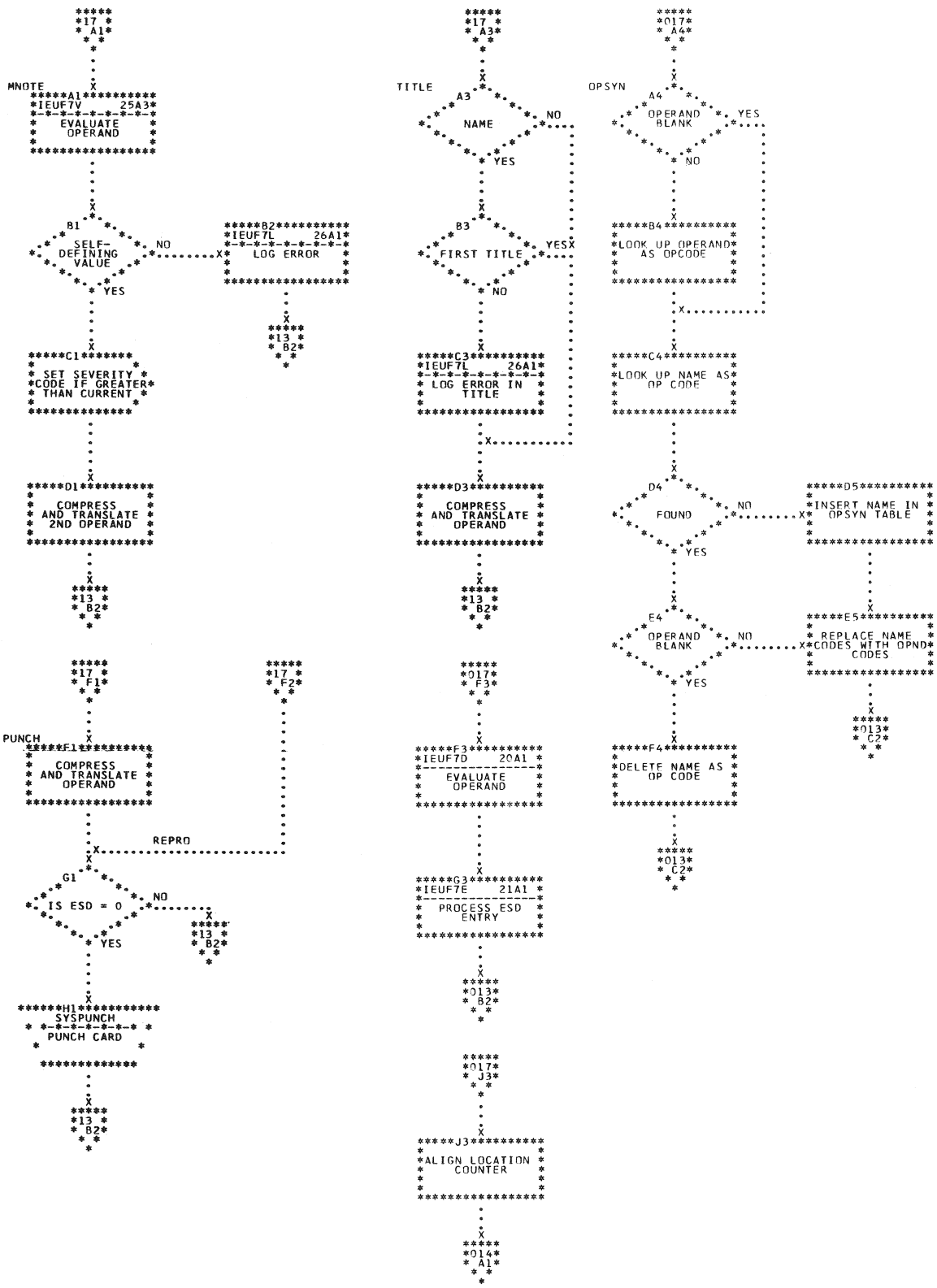


Chart 17. IEUF7C - Phase F7 Main Line Control (5 of 6)

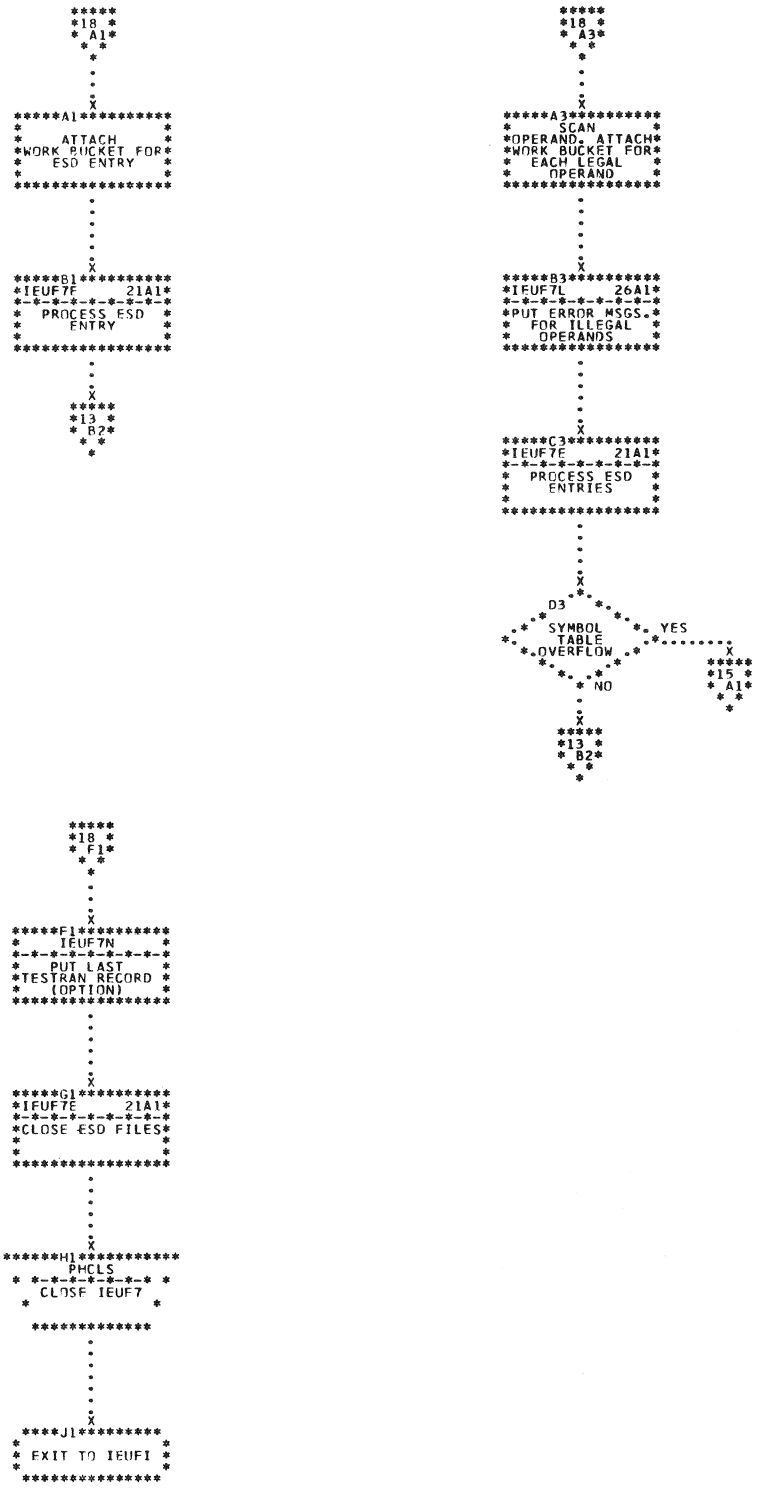


Chart 18. IEUF7C - Phase F7 Main Line Control (6 of 6)

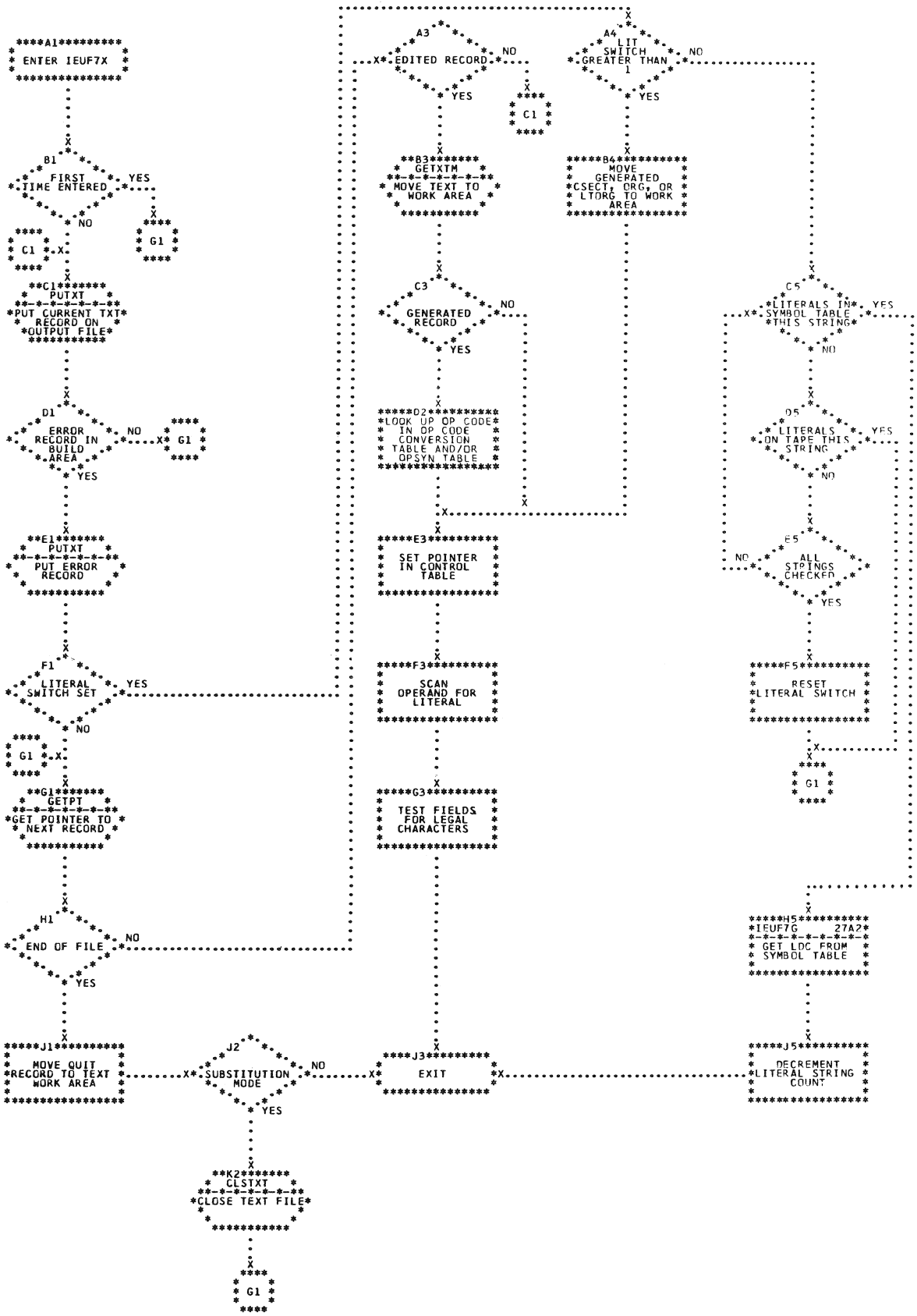


Chart 19. IEUF7X - Phase F7 Get Statement

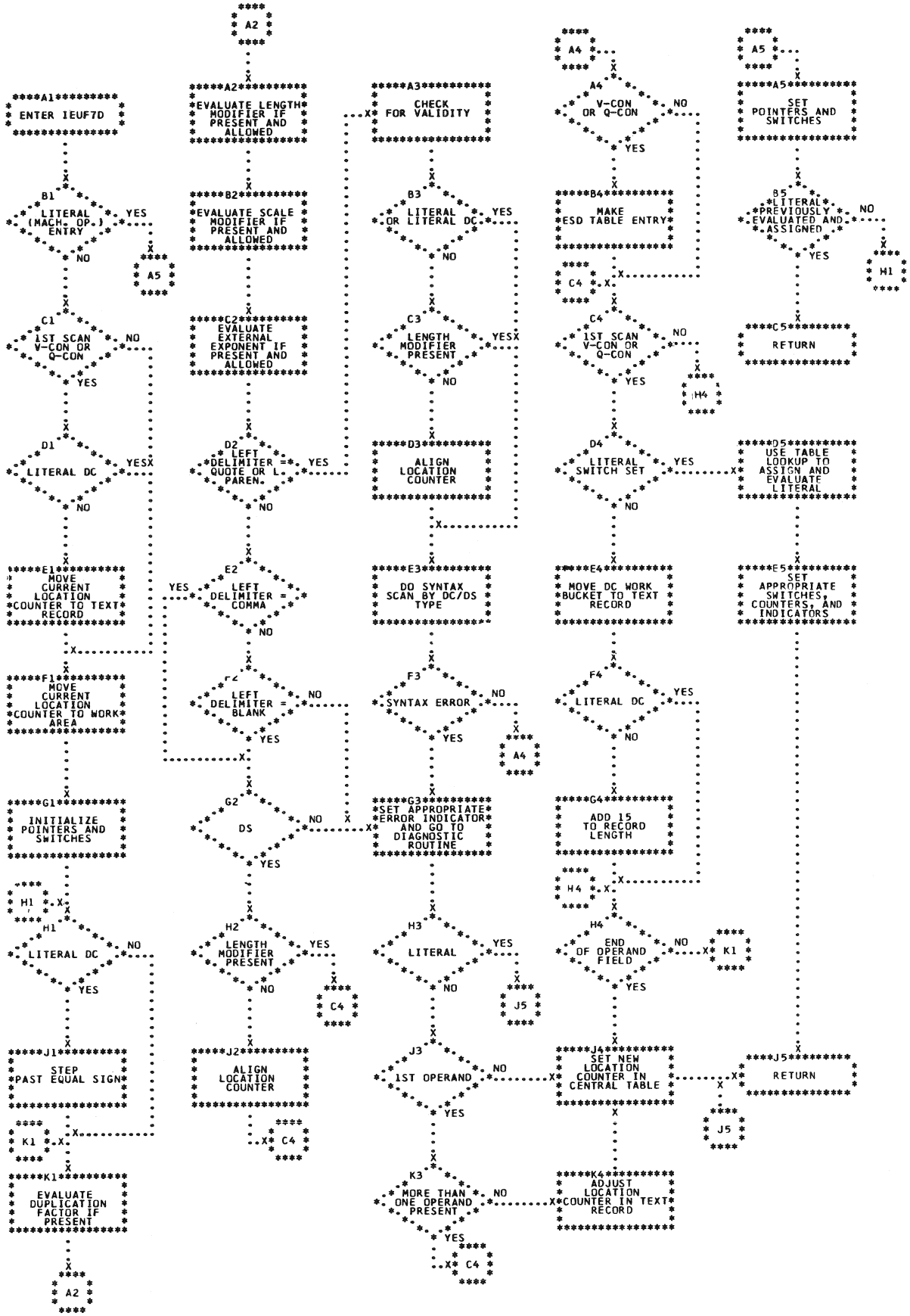


Chart 20. IEUF7D - Phase F7 DC Evaluation

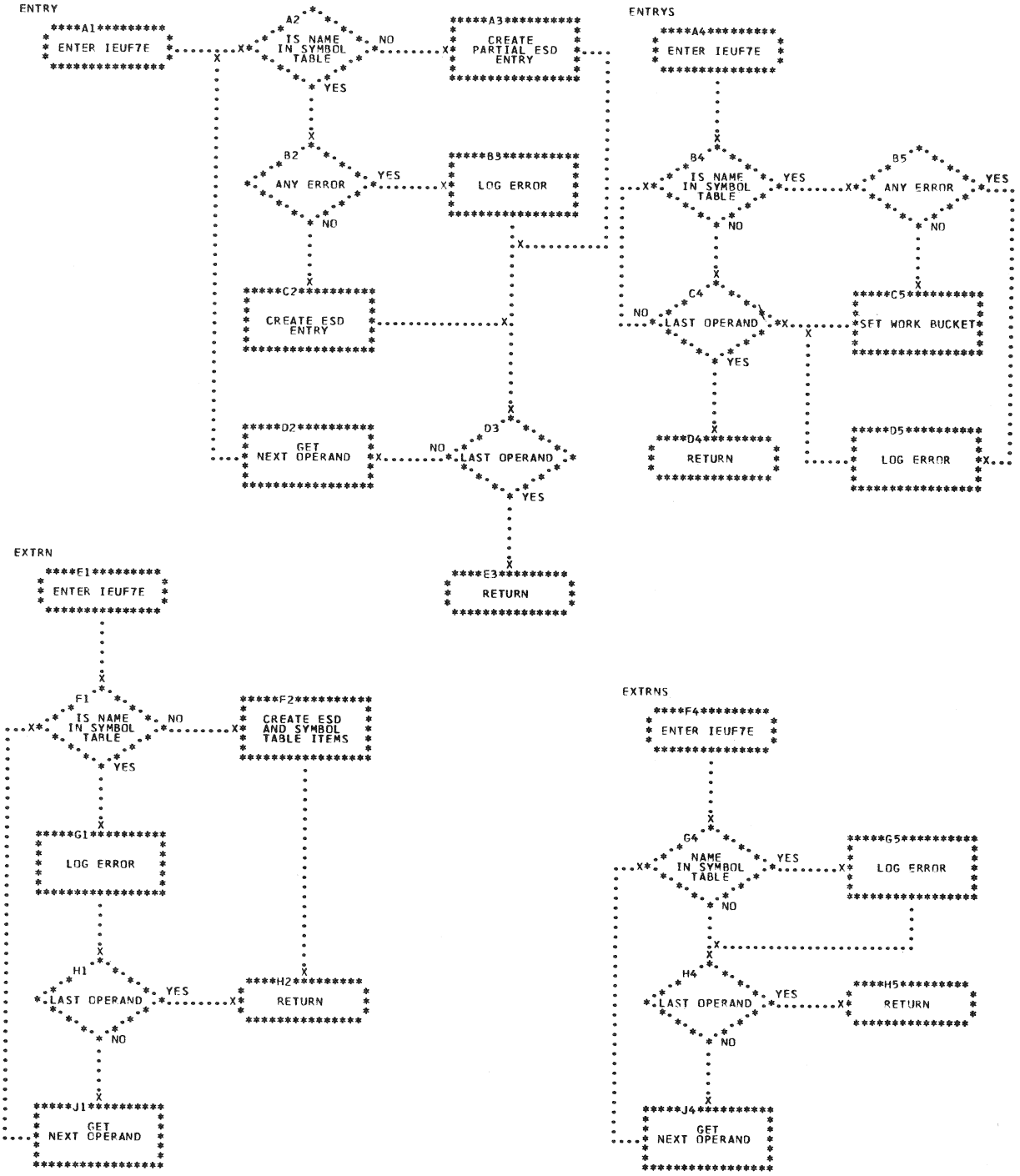


Chart 21. IEUF7E - Phase F7 ESD Routine (1 of 3)

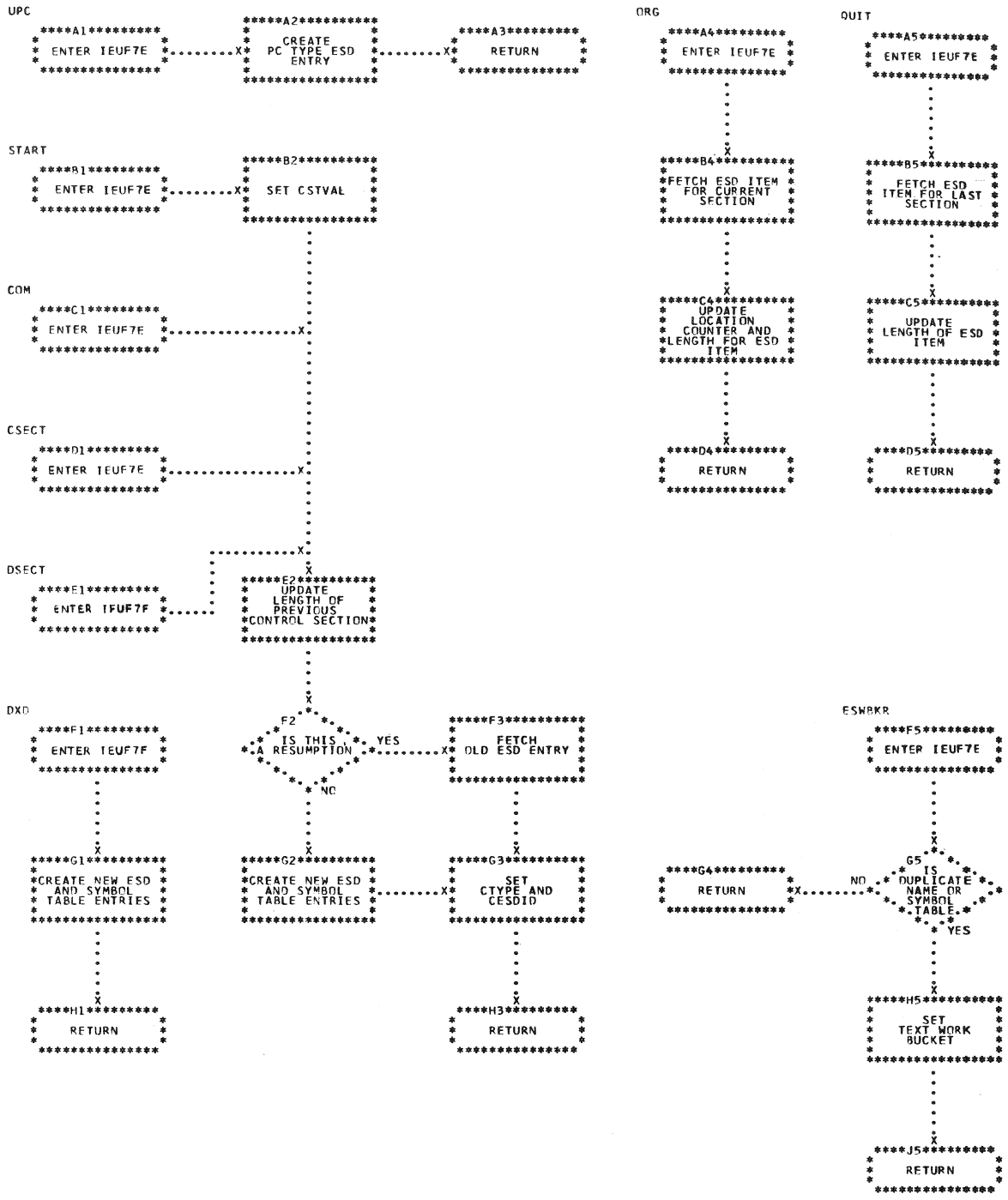
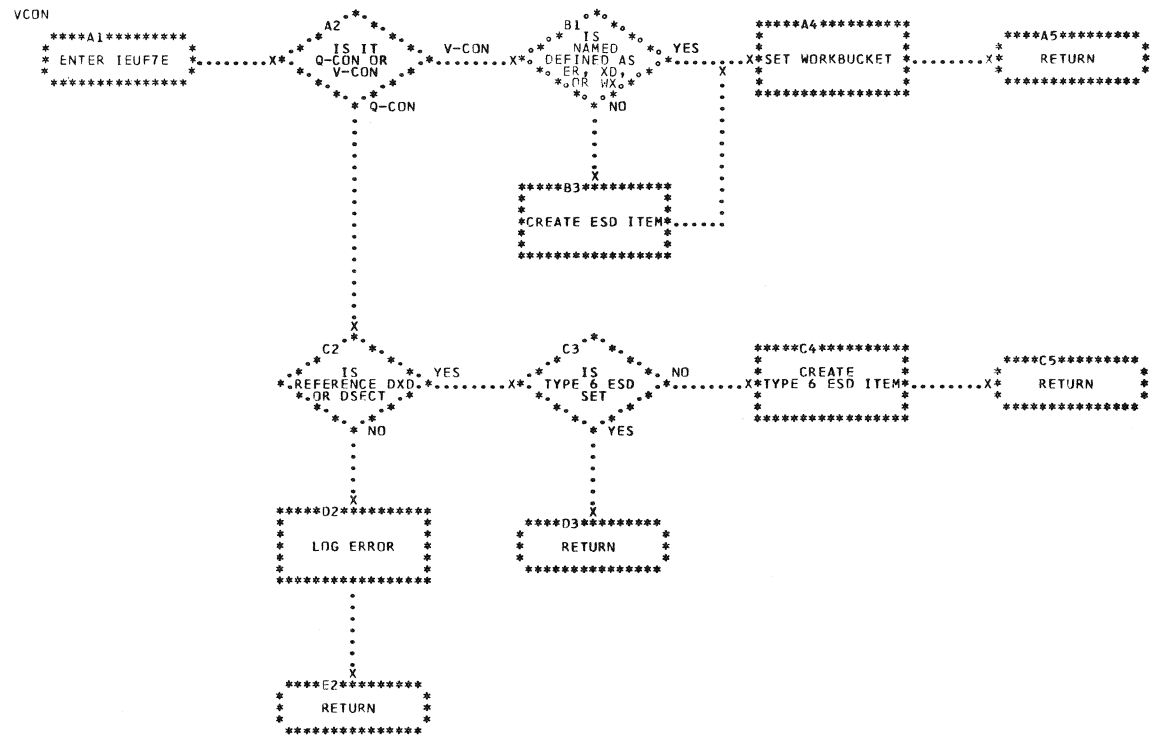


Chart 22. IEUF7E - Phase F7 ESD Routine (2 of 3)



• Chart 23. IEUF7E - Phase F7 ESD Routine (3 of 3)

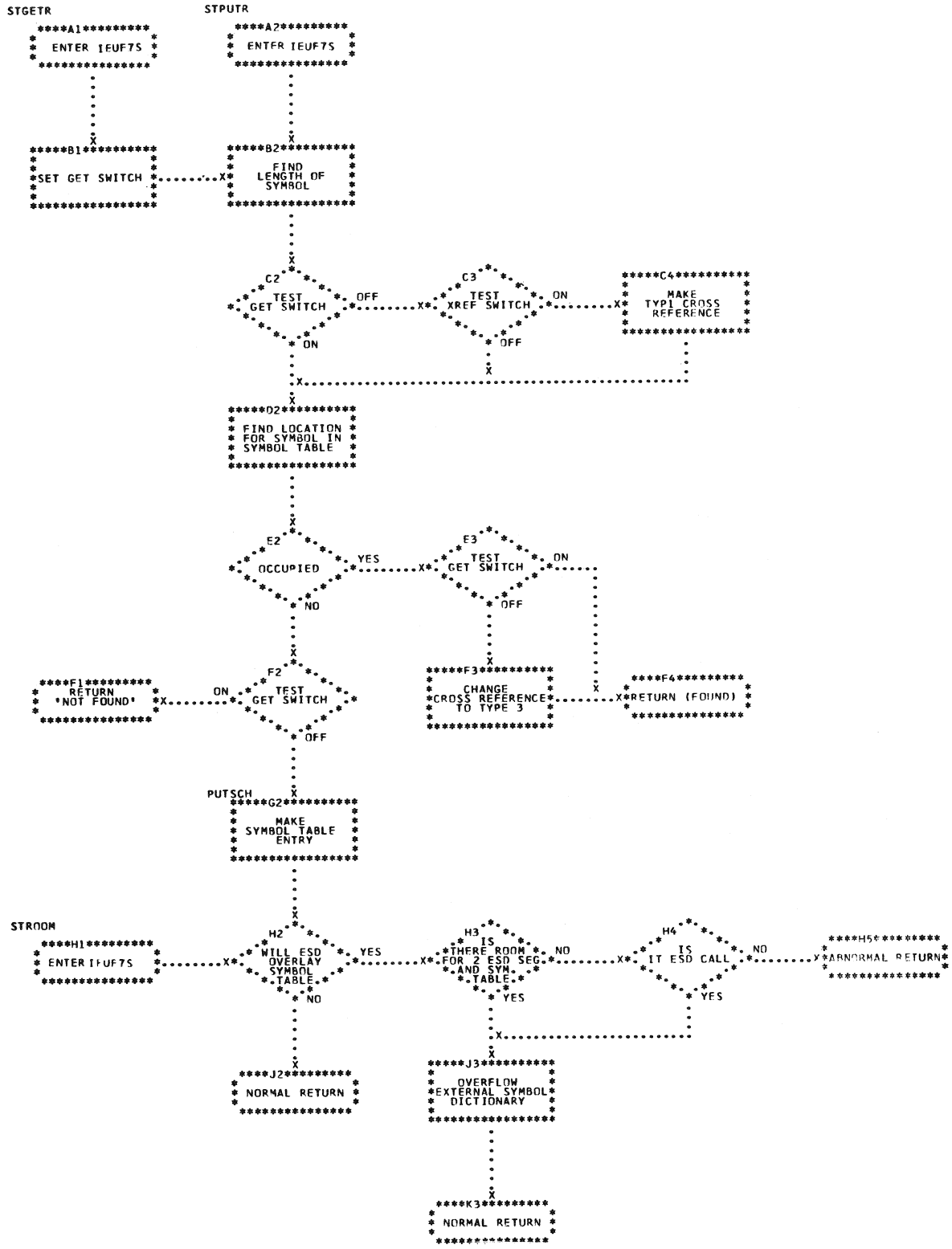


Chart 24. IEUF7S - Phase F7 Symbol Table Routine

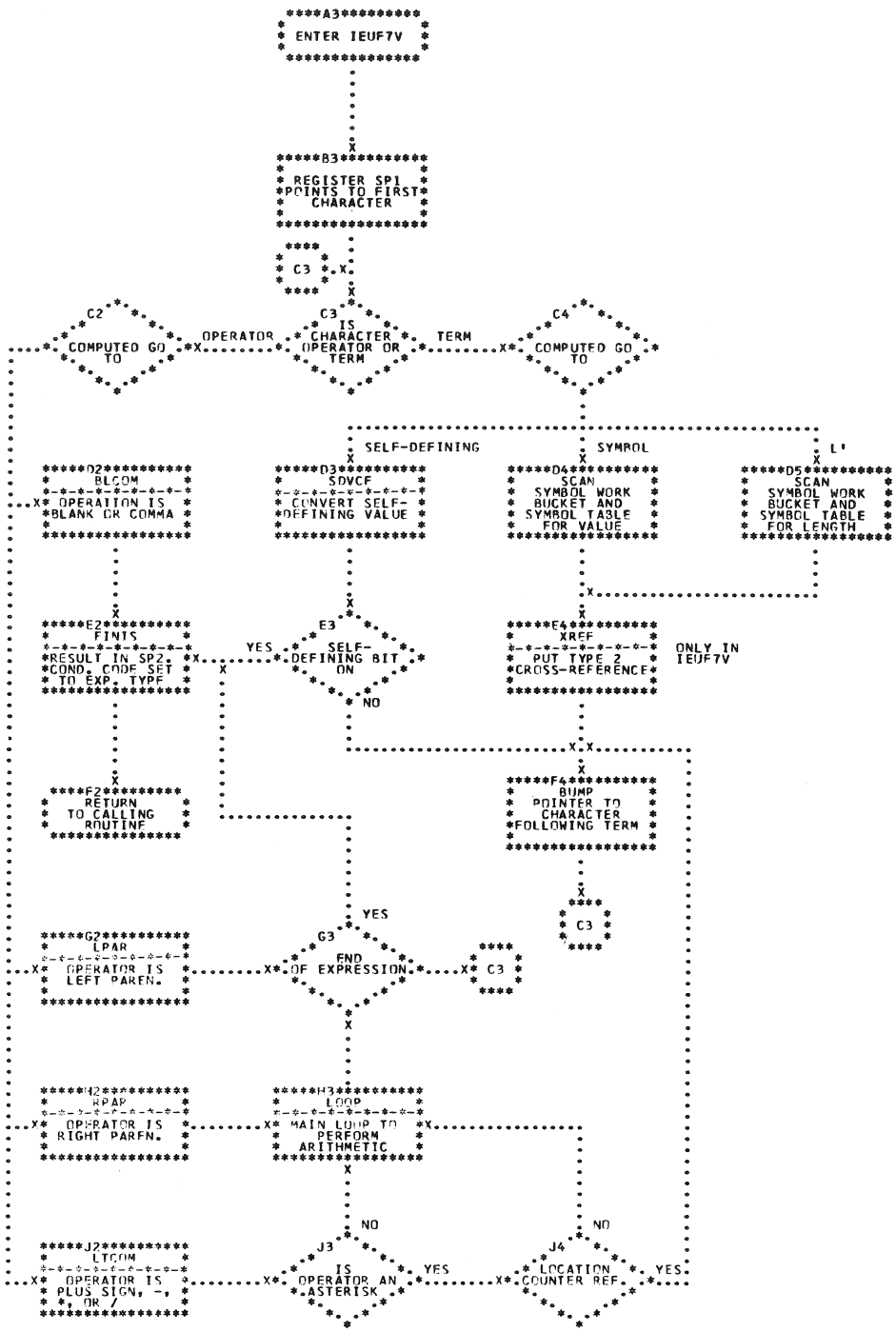


Chart 25. IEUF7V - Phase F7 Expression Evaluation

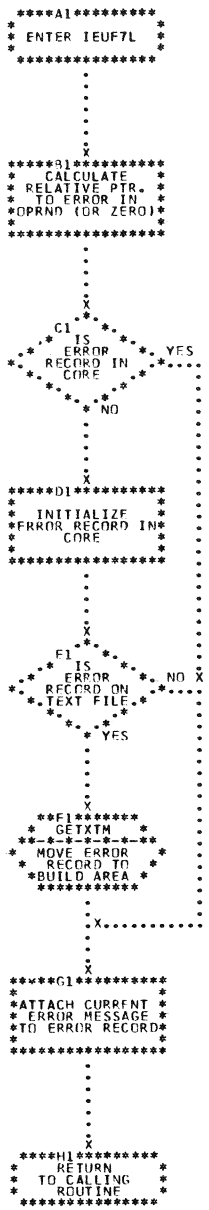


Chart 26. IEUF7L - Phase F7 Log Error Routine

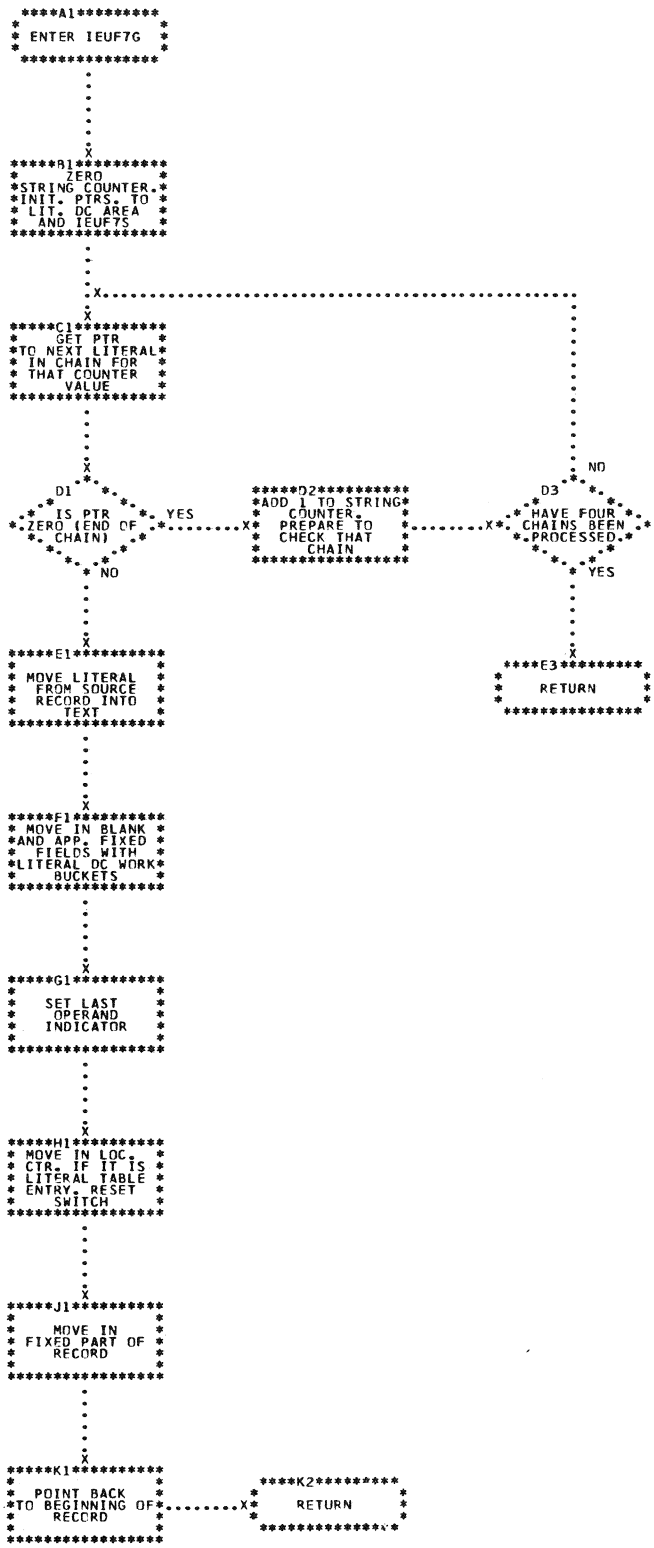


Chart 27. IEUF7G - Phase F7 DC Get Routine

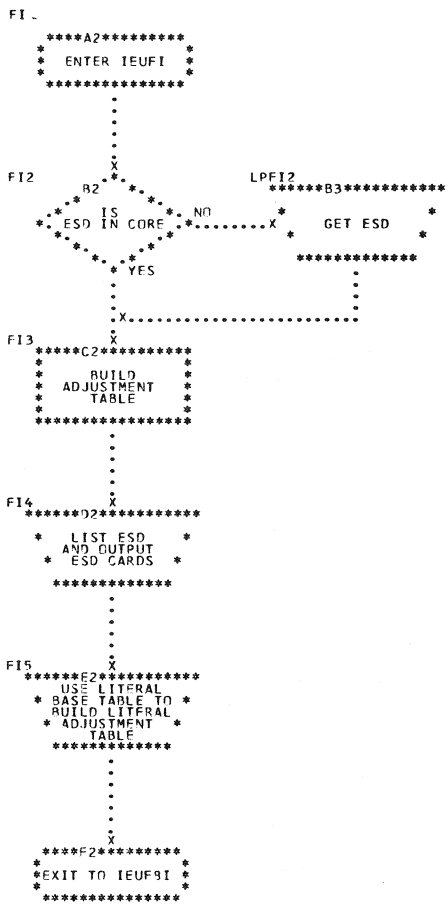


Chart 28. IEUFI - Phase F Interlude

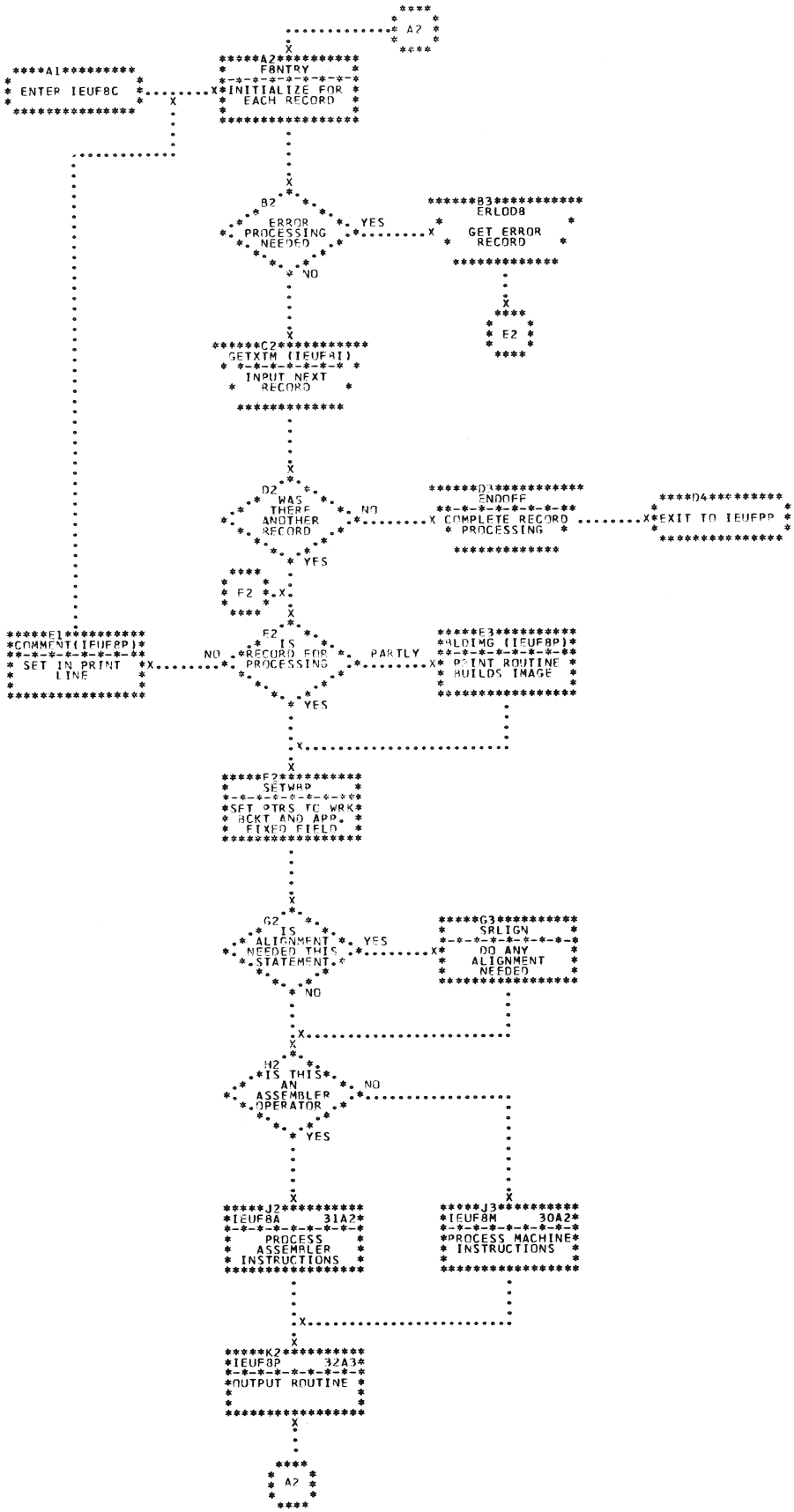


Chart 29. IEUF8C - Phase F8 Main Line Control

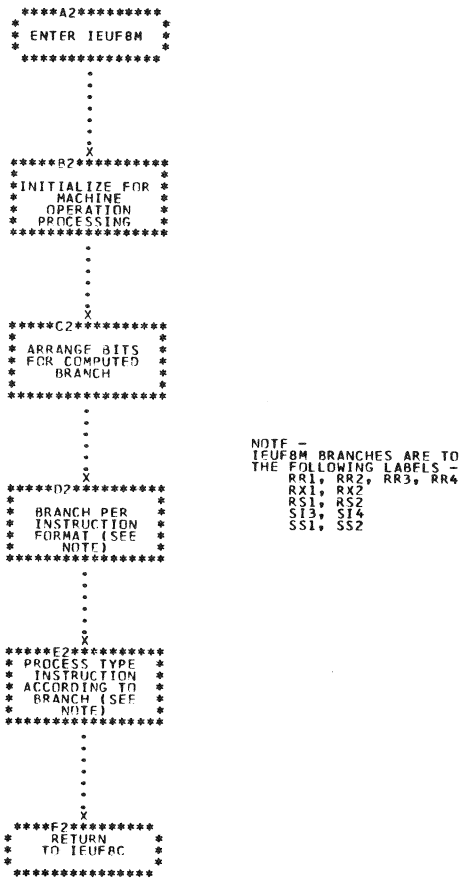


Chart 30. IEUF8M - Phase F8 Machine Operation Processor

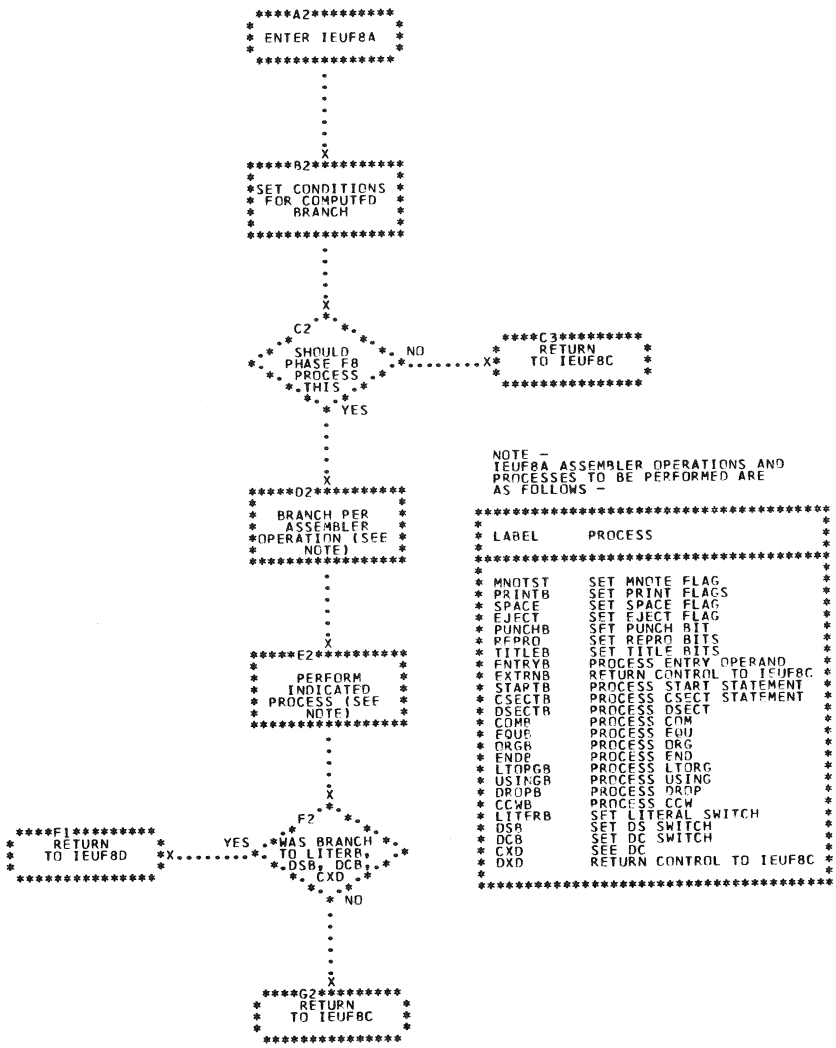


Chart 31. IEUF8A - Phase F8 Assembler Operation Processor

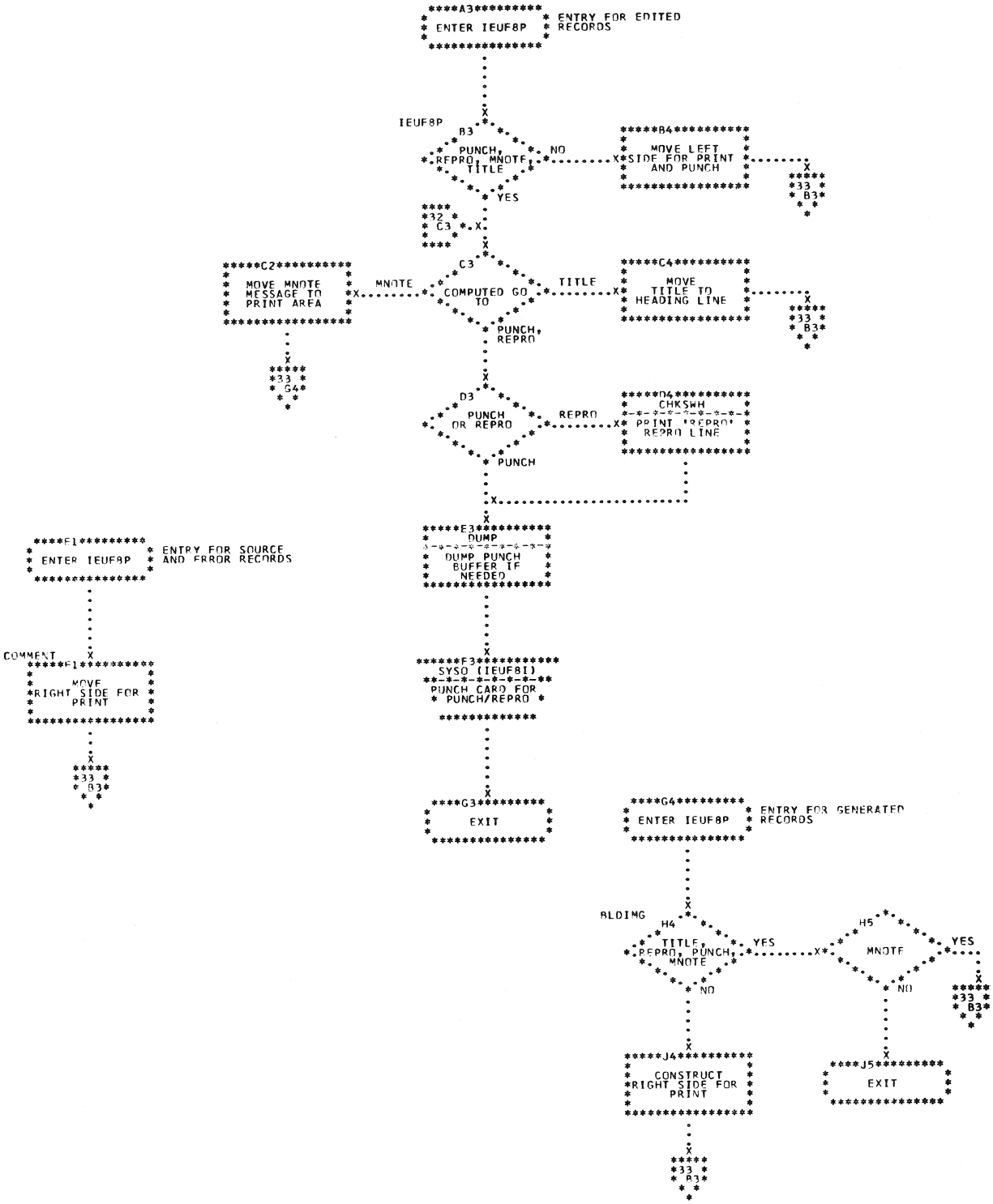


Chart 32. IEUF8P - Phase F8 Print Routine (1 of 2)

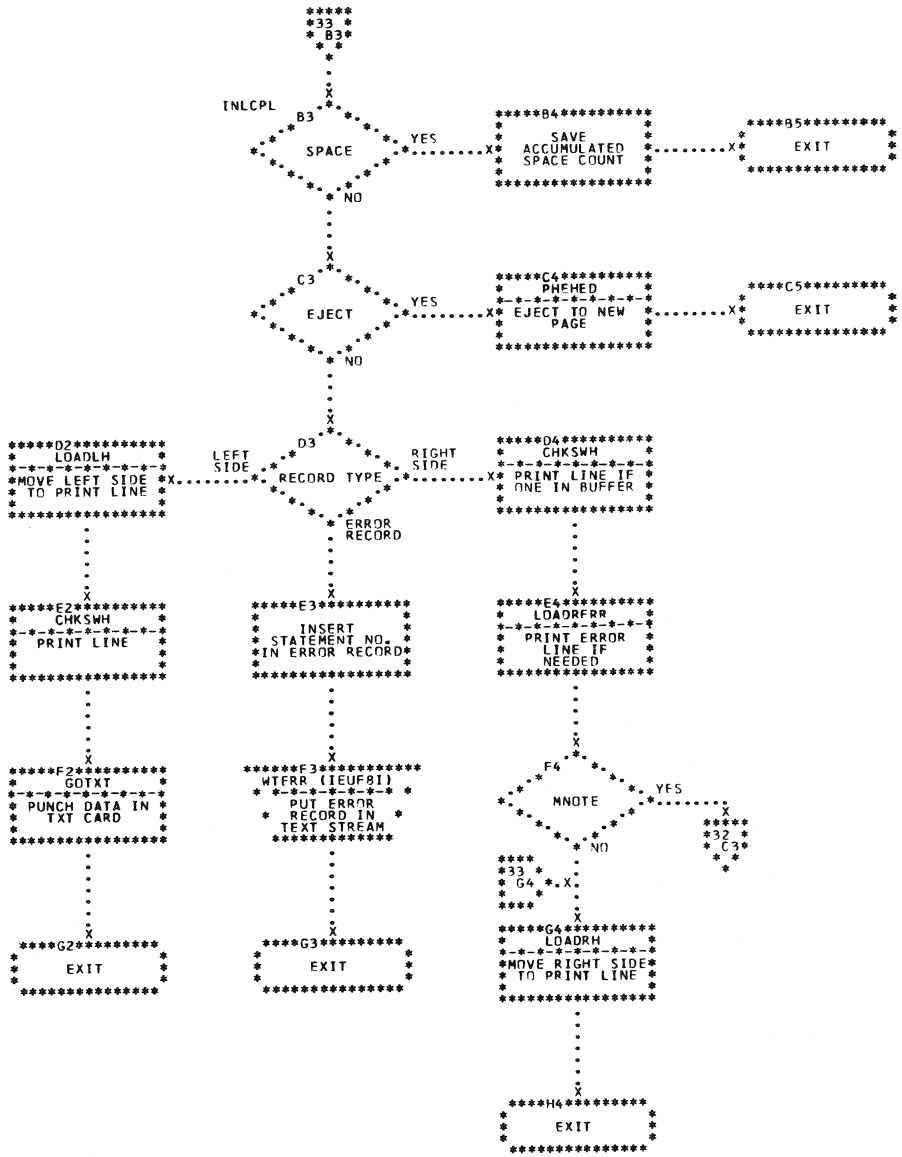


Chart 33. IEUF8P - Phase F8 Print Routine (2 of 2)

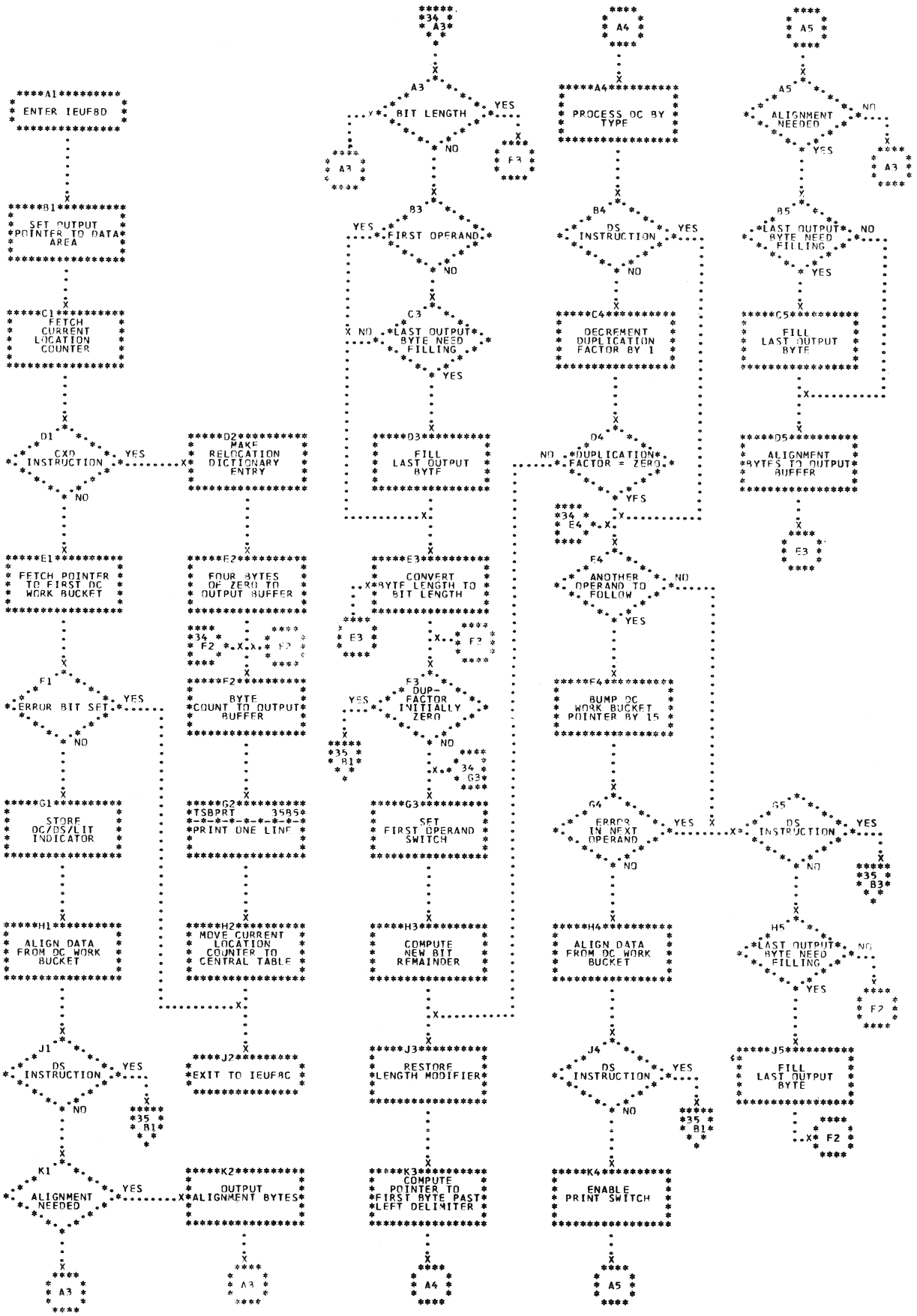


Chart 34. IEUF8D - Phase F8 DC Evaluation (1 of 2)

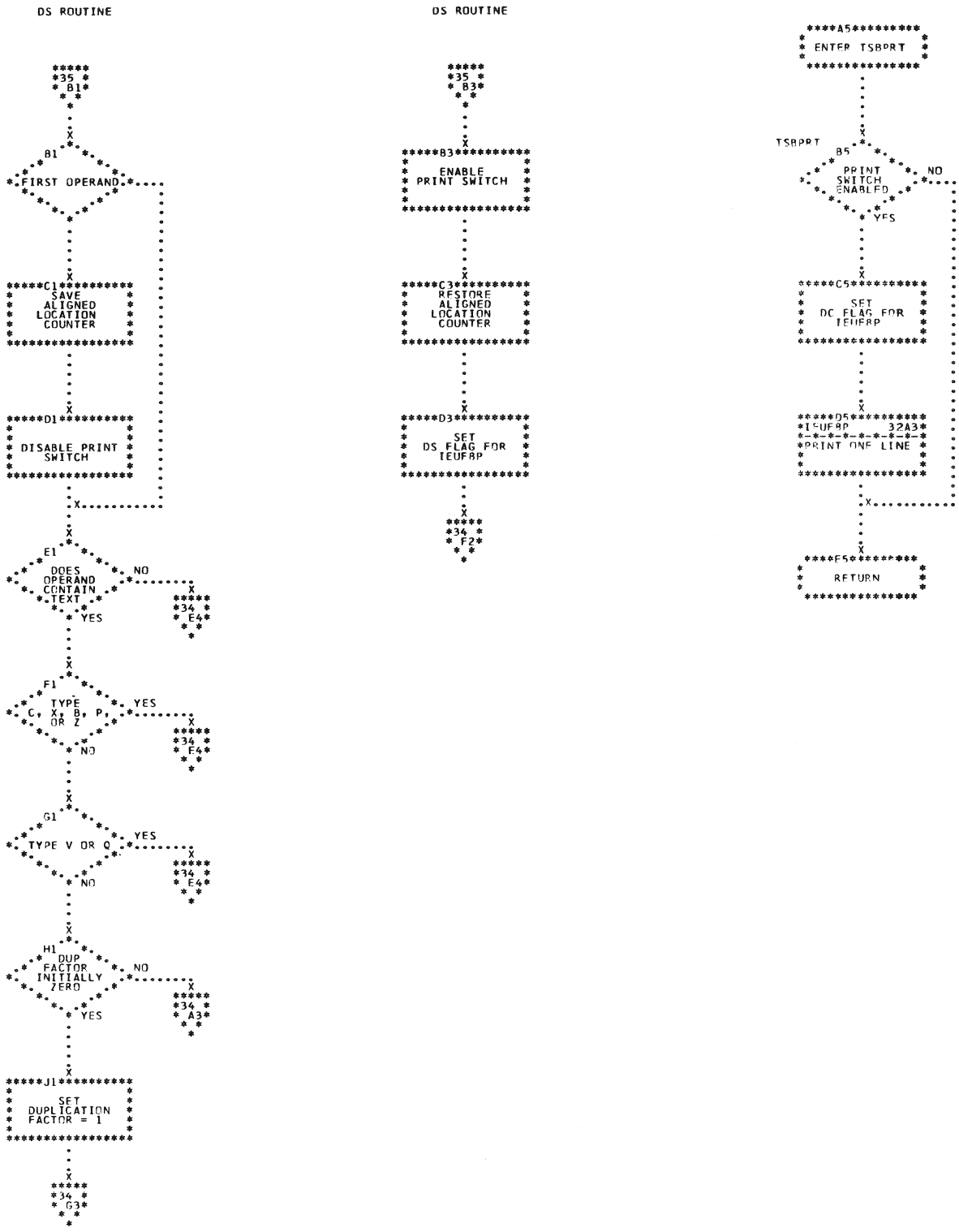


Chart 35. IEUF8D - Phase F8 DC Evaluation (2 of 2)

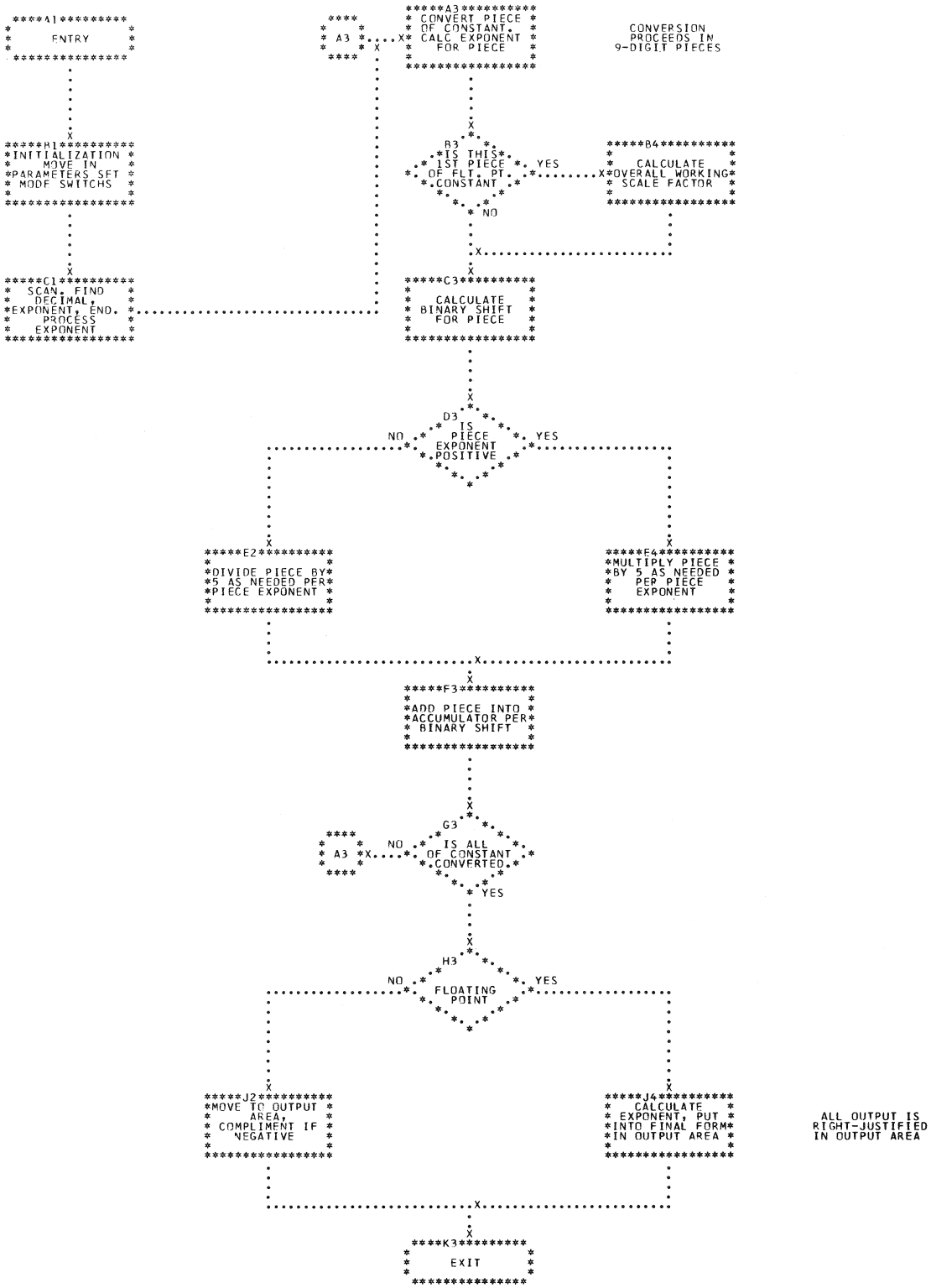


Chart 36. IEUF8N - Phase F8 Floating and Fixed-Point Conversion

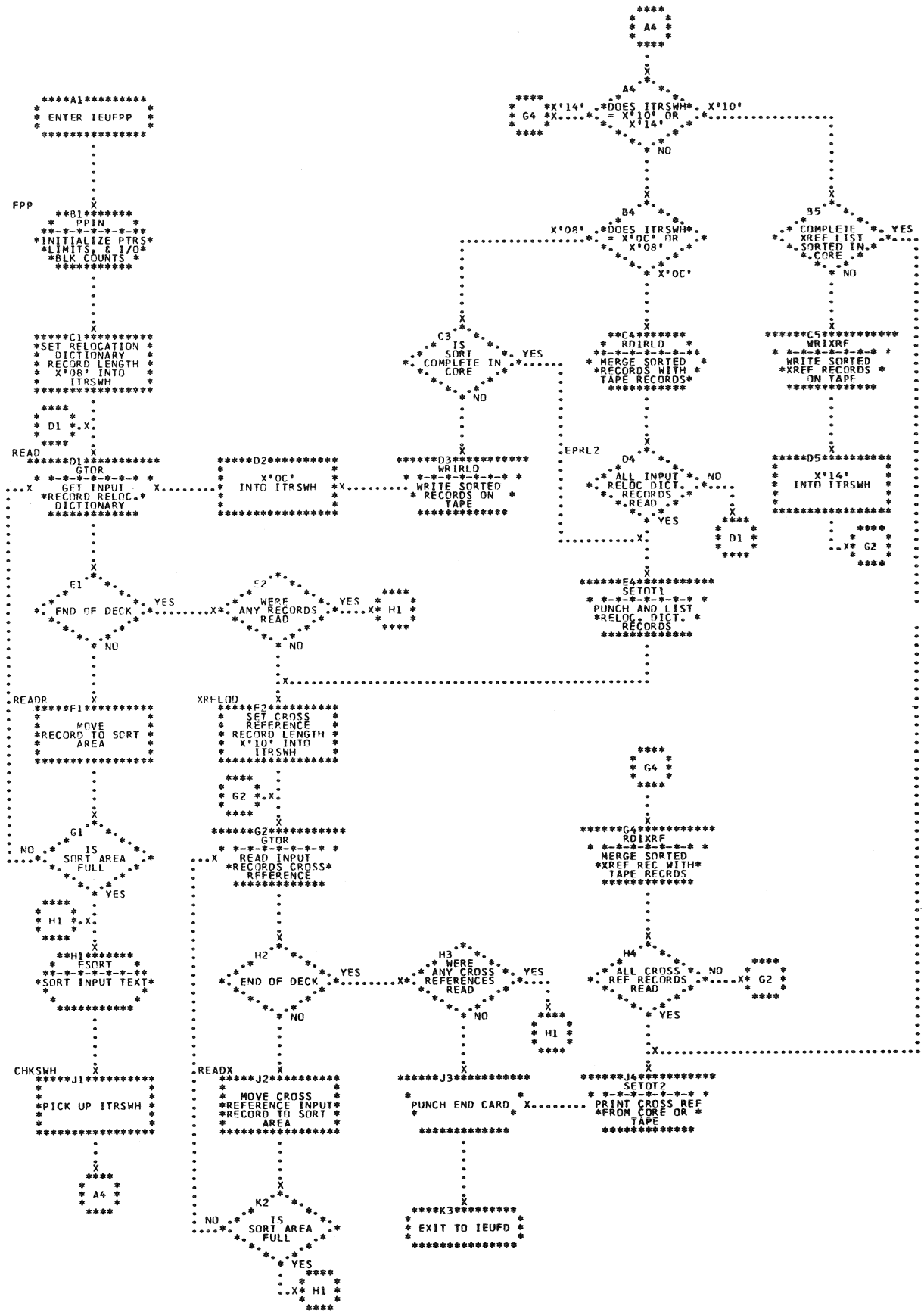
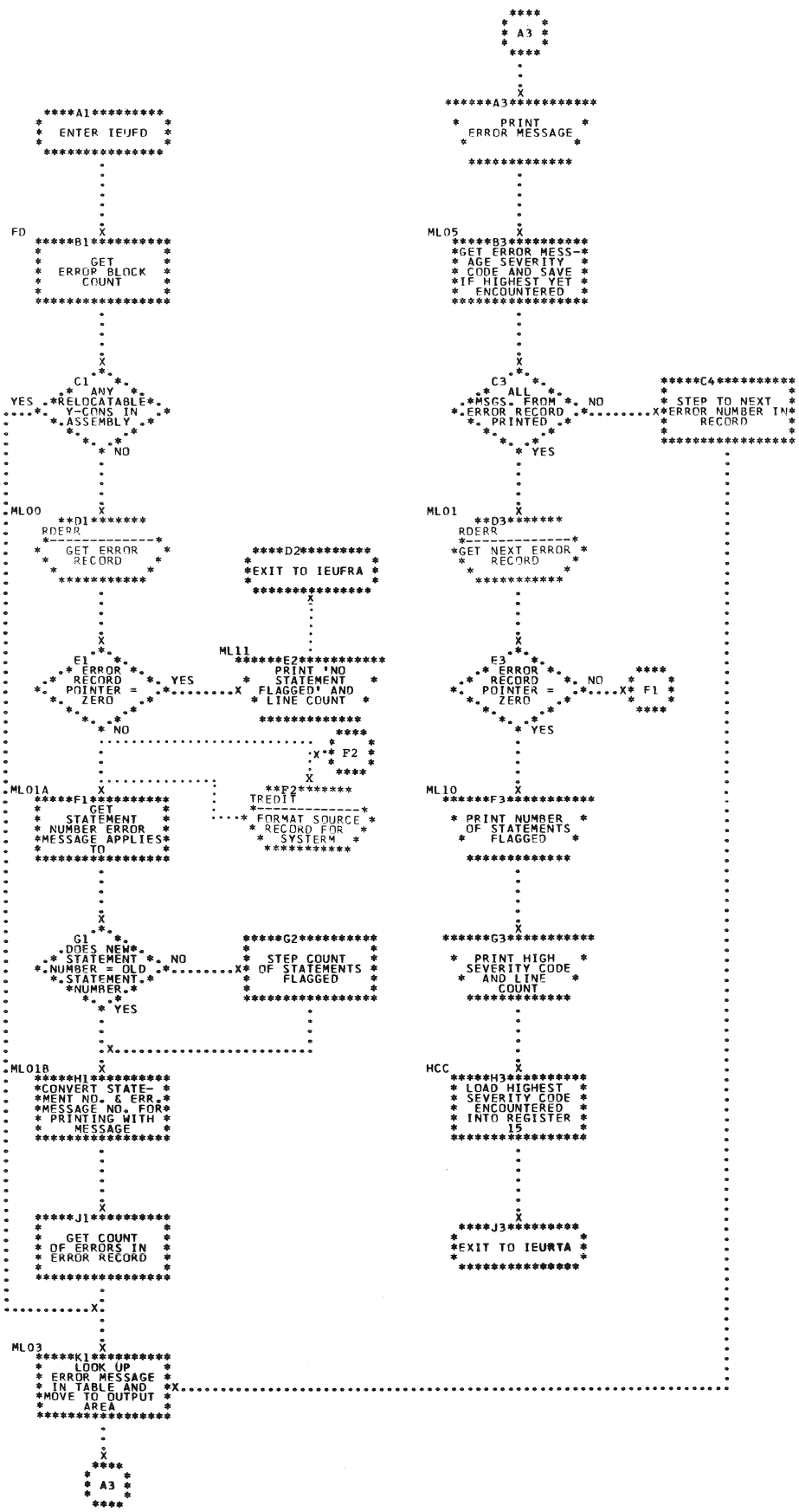


Chart 37. IEUFPP - Phase FPP Post Processor



• Chart 38. IEUFD - Phase FPP Diagnostic

OPTIONS

The programmer may specify the options listed in Figure 17 in the PARM= field of the EXEC statement. The options that are underlined in the figure are the default values that are assumed by the assembler if no value is specified in the EXEC card.

The options are defined as follows:

DECK -- The object module is placed on the device specified in the SYSPUNCH DD statement.

LOAD -- The object module is placed on the device specified in the SYSGO DD statement.

Specification of the parameter LOAD causes object output to be written on a data set with ddname SYSGO. This action occurs independently of the output on SYSPUNCH caused by the parameter DECK. The output on SYSGO and SYSPUNCH is identical except that SYSPUNCH is closed with a disposition of LEAVE, and SYSGO is closed with a disposition of REREAD.

LIST -- An assembler listing is produced.

TEST -- The object module (if produced) contains the special source symbol table required by the test translator (TESTRAN) routines.

XREF -- The assembler produces a cross-reference table of symbols as part of the listing.

RENT -- The assembler checks to see if the user's code is reentrant.

The prefix NO is used with the above options to indicate that the option is not wanted.

NOALGN -- The assembler supresses the diagnostic message IEU033 ALIGNMENT ERROR if fixed point, floating point, or logical data referenced by an instruction operand is not aligned on the proper boundary. The message will be produced, however, for references to instructions (e.g., by a branch) which are not aligned on the proper (halfword) boundary.

ALGN -- The assembler does not supress the alignment error diagnostic message.

OS -- The assembler will have complete Operating System Assembler F capability.

DOS -- The assembler will behave like Disk Operating System (DOS) Assemblers D and F. Anything defined in either of these assemblers with the exception of &SYSPARM will be accepted. CXD, DXD and OPSYN will be treated as undefined Q-type DC and DS statements and RLDs will appear in the Relocation Dictionary in order of their occurence (unsorted). DOS is incompatible with LOAD, TEST, and NOALGN. If any of these options are specified with DOS, the assembler generates a diagnostic message and uses NOLOAD, NOTEST, and ALGN.

If contradictory options are entered, e.g., LIST, NOLIST, the rightmost option, NOLIST, is used.

LINECNT=nn -- This parameter specifies the number of lines to be printed between headings in the listing. The permissible range is 01 to 99 lines.

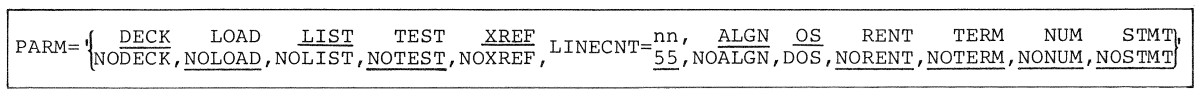
TERM -- The assembler writes diagnostic information on the SYSTERM data set. Options NUM and STMT can be specified only if TERM is used.

NUM -- The line number field (columns 73-80) is written on SYSTERM for statements for which diagnostic information is written on SYSTERM. This option is valid only in connection with TERM.

STMT -- Statement number will be written on SYSTERM for statements that are flagged by the assembler. STMT is valid only in connection with TERM.

Note 1: If option NUM or STMT is used together with NOTERM a diagnostic message (IEU078) will be generated.

Note 2: If option NOTERM is used for an assembly, NONUM and NOSTMT will not be listed after *OPTIONS IN EFFECT* in the diagnostic section of the listing.



• Figure 18. Assembler Options

APPENDIX B. CONTROL PROGRAM SERVICES

GENERAL

The control program services (macros) utilized during operation by the assembler are described briefly in this appendix. Detailed information is given in the Supervisor and Data Management Macro Instructions publication.

DCB

This macro interfaces with the control program. Information required to process a data set is presented by the assembler in this instruction.

GET

This macro is used to read SYSIN.

PUT

This macro is used to write SYSPRINT, SYSPUNCH, and SYSGO.

READ

This macro is used to read SYSLIB, SYSUT1, SYSUT2, and SYSUT3. READ retrieves the requested block from the input data set and places it in a main storage area.

WRITE

This macro is used to write SYSUT1, SYSUT2, and SYSUT3. Write transfers a block from the user's main storage area to a physically sequential data set.

CHECK

This macro checks the validity of READ or WRITE. It waits, if necessary, for the completion of a read or write operation and detects errors and exceptional conditions.

NOTE

This macro records the position of an element in a data set by requesting the relative position within a volume of the block just read or written. This allows the direct retrieval of the block at a later time.

POINT

This macro positions a data set for a READ (POINTR) or a WRITE (POINTW) from a NOTEd location. 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.

GETMAIN

This macro acquires main storage for internal tables, buffers, etc., by requesting the supervisor to allocate one or more areas of main storage for assembler use.

FREEMAIN

This macro releases one or more areas of main storage previously acquired through one or more GETMAIN macro instructions.

FIND

This macro locates a partitioned data set member by placing the address of its first block in the indicated data control block.

OPEN

This macro is used to initialize one or more data control blocks so that their associated data sets can be processed.

CLOSE

This macro disconnects one or more data sets from a program by closing one or more data control blocks.

CLOSE (TYPE T = TCLOSE)

This form of the CLOSE macro instruction can be used to temporarily disconnect one or more data sets from a program. After repositioning to a starting point, as specified in the macro, the data set may be processed again.

LINK

This macro is used to link to a program segment. LINK passes control from one load module to a specified entry point in another load module, keeping both modules resident in main storage.

XCTL

This macro transfers control from one load module to another. The main storage area occupied by the module issuing the XCTL instruction is freed for other uses.

RETURN

This macro returns control from a LINKed program segment. It indicates normal termination of a task and returns control to a higher level program or task, or to the control program.

APPENDIX C. SYSTEM OVERHEAD

System Overhead consists of those features in the control program which the assembler uses once regardless of source program size. Included in this estimate are the following:

Function	Estimated Time (secs.)
2 separate OPEN	5.76 ± 20%
2 separate CLOSE	2.25 ± 20%
3 separate LINK (.4 sec. each)	1.20 ± 20%
6 separate XCTL (.4 sec. each)	2.40 ± 20%
6 separate FIND (.3 sec. each)	1.80 ± 20%
6 separate TCLOSE (.2 sec. each)	1.20 ± 20%
Scheduler Time	10.00 ± 20%
System Overhead	24.61
Maximum System Overhead	29.53

APPENDIX D. INTERNAL ASSEMBLER INSTRUCTION CODES

Mnemonic	Decimal Value	Hexadecimal Value
GBLA	0	0
GBLB	1	1
GBLC	2	2
LCLA	3	3
LCLB	4	4
LCLC	5	5
SETA	6	6
SETB	7	7
SETC	8	8
AIF	9	9
AGO	10	A
ANOP	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
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
Literal DC	37	25
DC	38	26
DS	39	27
CCW	40	28
CNOP	41	29
DXD	43	2B
CXD	44	2C
OPSYN	45	2D
WXTRN	46	2E

APPENDIX E. TRANSLATE TABLE

All characters in source statements are translated to an internal hexadecimal coding. 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 external code before output. Bit configurations not representing Operating System/360 Assembler Language characters, e.g., valid overpunch characters in fields PUNCHed or REPROed, are not affected by the translation. (They are translated into themselves.) See Table 12.

Application of the translate table also allows the user to assemble programs written in other than System/360 Operating System Assembler Language by providing a different translate table for the conversion.

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

Table 12. Translate Table

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		

PHASE F3 SWITCHES

MISWIT

Bit No.	Hex. Sw.	
0	X'80'	Not used
1	X'40'	0 - Do another pass on macro instruction operands 1 - 2nd pass completed on macro instruction operands
2	X'20'	1 - Macro instruction being processed
3	X'10'	1 - Entry is a sublist
4	X'08'	0 - Entry to be made from prototype 1 - Entry to be made from macro instruction
5	X'04'	1 - Nest aborted; no room to store parameter pointers
6	X'02'	1 - Macro aborted during build of parameter table
7	X'01'	1 - Max. record exceeded (used by WRITE routine)

SWITCH

Bit No.	Hex. Sw.	
2	X'20'	0 - NOTE in IOMAC Routine 1 - No need to NOTE in IOMAC routine
4	X'08'	0 - Use PNTR in IOMAC routine 1 - Use PNTW in IOMAC routine
7	X'01'	1 - End of Expr. 1 or Expr. 2 of substring has been reached

NESTSW

Bit No.	Hex. Sw.	
6	X'02'	1 - Set when a new block has been read when processing a macro instruction.
7	X'01'	1 - Macro instruction mode - set when a macro instruction is encountered so that nesting may be recognized.

SUBSW

Bit No.	Hex. Sw.	
6	X'02'	1 - Set when MODESW has been saved. Char. or arith. mode must be saved when a subscripted left parentheses is encountered so that this mode may be restored after the subscript dimension is computed.
7	X'01'	This must always be initialized to the value of 1, used when original MODESW is restored.

MODESW

Bit No.	Hex. Sw.	
0	X'80'	1 - SYSLIST to provide alternate to symbolic parameters
1	X'40'	1 - 2 expressions in SYSLIST
2	X'20'	1 - Concatenation has occurred in string area
3	X'10'	1 - Error switch in substring routine - signal to use null string later on
4	X'08'	1 - First time switch - set after first char. string has been placed in string area
5	X'04'	1 - Substring mode
6		Not available for use
7	X'01'	0 - arithmetic expr. mode 1 - character expr. mode

PHASE F7, FI, F8 AND FPP SWITCHES

Name	Comment	Name	Comment
CTLOC	Current Location Counter	CTPGLNCT	Page Line Count
CTSEQN	Current Statement Sequence Number	CTMRSRTN	MRS Return
CTLEN	Current Statement Length	CTZERO	Two Full Words of Zeroes
CTITLE	First Title Name, Opnd. Len, Opnd. Ptr.	CTWORK	256 Byte Work Area
STVALU	Value For STPUT Entries	CTONWP	Next Write Pointer On OVF1
CPRIME	Prime Divisor For Symbol Table	CTRXF	First XRF Block PTR On OVF1
CSTVAL	Value From START card	CTRLBT	First LBT Block PTR On OVF1
CTXLEN	Text Block Length	CTRERR	First Error Block (PH8)
CNOESD	Number of ESDs	CTCXRF	XRF Block Count
CENTCT	Number of Entries	CTCLBT	LBT Block Count
CLASID	Last ID	CTCRLB	RLD Block Count
CTNDID	Next DSECT ID	CTCERR	Error Block Count (PH8)
CESDNO	Current ESD Number	CTCLAT	LAT Block Count On OVF2
CSGCTR	ESD Resident Segment Counter	CTLALN	LAT Length Indicator
CPCNO	Private Code ESD Number	CTLITA	Current Literal Pool String Lengths
CCMNO	Common ESD Number	CTLITB	Current Literal Pool String Counts
STLONG	Length Attribute For STPUT Entries	CTXSAV	
ESSGSZ	ESD Segment Size	CTFSTN	ESD No. of First CSECT
CESDID	Current ESD ID	CTDATE	Date For Listing
CTPCSW	Private Code Switch	CTLINECT	Print-Line Count
CTCMSW	Common Switch	CADJTB	Adjustment Table Base
CFSTID	First CSECT ID	RR2SWH	RR2 Instruction Type Switch
CTYPE	Current CSECT Type	ERSWH	ERROR Switch
CTLIT2	LTORG Or END Card Switch	SPACSW	SPACE Switch
ESDID	Assigned ESD ID	EJCTSW	EJECT Switch
ADJCOD	Adjective Code	REPSW	REPRO Switch
CTALIN	Alignment Code 0-B, 1-H, 3-F, 7-D	CCRDCT	Card Count
CTITSW	Iteration Switch	CTLATL	Literal Adj. Tab.- Last Byte + 1
CTPDS1	Defined Symbols Req. For IEUF7V	ENDSWH	END Switch
CTCLSI	First Pass Indicator	FBOPRN	Operand Pointer
CTLITI	Literal Pool Complete During Subst.	CTLATB	
CTERRI	Error Record Indicator	F8CADJ	Current Adjustment
CTPH7C	Phase F7 Complete Indicator	ALIGN4	For Aligning
CTSYMF	Symbol Table Full Indicator	F8ALLB	Full Word Of Bits
CTPCHI	Punch Option Indicator	F83BYT	3 Bytes Of Bits, Low Order
CTCGOI	CGO Option Indicator	F82BYT	2 Bytes Of Bits, Low Order
CTITLI	First Title Processed Indicator	F81BYT	1 Byte Of Bits, Low Order
CTLSTI	List Option Indicator	F8PON	Print Option ON-OFF Switch
CTGENI	List Gen. Option Indicator	F8PGEN	Print Option GEN-NOGEN Switch
CTERLI	List Error Option Indicator	F8PDAT	Print Option DATA-NODATA Switch
CTXRFI	X-Ref. Option Indicator	F8ZERO	One Full Word Of Zero
CTTSTI	TESTRAN Option Indicator		
CTTRMI	TERM Option Indicator		
CTSTMI	STMT Option Indicator		
CTNUMI	NUM Option Indicator		
CTSDVI	Self Defining Value Indicator	F8INST	16 Byte Instruction Bldg. Area
CTLCRI	Location Counter Reference Indicator	F8ZRO	One Full Word Of Zero
CTMODE	Mode Indicator	PYRSW	
CBDNO	Blank DSECT ESD No	F8YDC	
CBDSW	Blank DSECT ID No	CTESRN	ESD Seg. Count

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 program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

Assembler Operation Code: A hexadecimal one-byte code assigned to all assembler instructions by programming systems for internal use.

Attributes: Characteristics of certain elements in statements processed by the assembler. There are six attributes: type, length, scaling, integer, count, and number. The macro generator processes all of them; the assembler portion, 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.

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

Conditional Assembly: The selective assembly of those source language statements that satisfy predetermined conditions, e.g., tests of values that may be defined, set, or changed during the course of the assembly procedure. The conditional assembly precedes the regular assembly procedure. Conditional assembly allows a programmer to specify assembler language statements which may or may not be assembled depending on conditions evaluated at assembly time.

Control Program: A collective term for the operation and resource controlling routines of the operating system.

Control Section: The smallest separately relocatable program unit, always loaded into a contiguous main storage area. A control section is an entity. Its name, if there is one, is defined by a CSECT or START statement.

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 containing information for access routines pertinent to data storage and retrieval.

Data Set: A named collection of data.

Device Independence: The ability to request input/output operations without regard to the characteristics of the input/output devices.

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: Part of an object or load module that identifies external names (control sections, ENTRY statements, common areas, and private codes) and external references (EXTRN statements and V-type address constants) occurring in the module.

External Symbol Dictionary Identifier (ESD-ID): A one-byte number identifying a control section or other external symbol dictionary entry.

Global Dictionary: A core storage resident table containing machine and assembler operation codes, macro mnemonics, and global variable symbols.

Global Variable Symbols: Global SET symbols (the only type of global variables) that communicate values between statements in one or more macro definitions and statements outside macro definitions.

Hashing: Generating an address between two limits by randomization.

Hash Table: A table, accessed through generated numbers (i.e., randomization), pointing to entries in a dictionary or table.

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

Linkage Editor: A program that produces a load module from object and/or load modules. The output load module is in a format suitable for loading and

execution under the control of the control program of the operating system.

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 relocatable and executable logical unit of coding. It is the output of the linkage editor in a format suitable for loading into main storage.

Local Dictionary: A table containing sequence symbols, ordinary symbols, local SET symbols, and macro instruction parameters.

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 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. 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.

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

Module: A logical unit of coding that performs a function or several related functions.
A source module is a set of source language statements prepared for input to a language translator.
An object module is the output of a language translator,(e.g., assembler). It is a machine language program in relocatable format. A load module is the output of the linkage editor. It is in relocatable and executable format.
A module is composed of one or more sections (see Control Section).

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

Operating System (IBM System/360 Operating System): A complex of processing programs operating under the supervision of the control program. The processing programs include language translators (e.g., the assembler), service programs (e.g., utilities), and problem programs. The operating system facilitates device independent operations.

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, dictionary, or data set entry.

Position Identifier: A two-byte value specifying the sign and external symbol dictionary identifier (ESD-ID) of the control section in which a relocatable constant occurs.

Prototype Statement: See Macro Instruction Prototype.

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

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

Relocation Identifier: A two-byte value specifying the sign and external symbol dictionary identifier (ESD-ID) of an item referenced by a relocatable constant.

Source Program: A series of statements in a a source language 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 System/360 Operating System, 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: Fields attached to certain types of records for holding internal information during processing.

INDEX

Indexes to program logic manuals are consolidated in the publication IBM System/360 Operating System Program Logic Manual Master Index, Order No. GY28-6717. For additional information about any subject listed below refer to other publications listed for the same subject in the Master Index.

IBM System/360 OS Assembler (F) (S360-21) Printed in U. S. A. GY26-3700-2

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Operating System
Assembler (F)
Program Logic Manual

Order No. GY26-3700-2

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

  As a reference source:

Very Good	Good	Fair	Poor	Very Poor
.....	.....	.....	.....	.....

  As a text:

Very Good	Good	Fair	Poor	Very Poor
.....	.....	.....	.....	.....

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

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

YOUR COMMENTS, PLEASE . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation

112 East Post Road

White Plains, N. Y. 10601

Attention: Department 813 L

Fold Fold

IBM

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

CUT ALONG THIS LINE

IBM System/360 OS Assembler (F) (S360-21) Printed in U. S. A. GY26-3700-2

IBM Technical Newsletter

File No. S360-21 (OS)

Base Publ. No. GY26-3700-2

This Newsletter No. GN33-8102

Date June 1, 1971

Previous Newsletter Nos. None

IBM SYSTEM/360 OPERATING SYSTEM ASSEMBLER (F)
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of release 20.1 of IBM System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below:

Front Cover,Preface

A change to the text or an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

Minor technical and editorial changes made to the abstract and edition notice.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Nordic Laboratory, Technical Communications, Box 962, S-181 09 Lidingö 9, Sweden

PRINTED IN U.S.A.