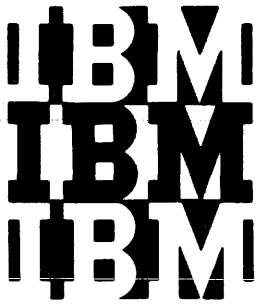
File No. S360-21
Form Y26-3716-0



# Program Logic

## IBM System/360 Disk Operating System Assembler [F]

**Program No. 360N-AS-466**

This publication describes the internal logic of the F (64K) Assembler for the IBM System/360 Disk Operating System. It is intended for use by persons involved in program maintenance and by system programmers who are altering the program design. Since program logic information is not necessary for the use and operation of the Assembler, distribution of this publication is limited to these people.

**Restricted Distribution**

## PREFACE

This manual describes the internal logic of the IBM System/360 Disk Operating System F Assembler. The introduction gives the purpose of the assembler and summarizes its organization, theory of operation, and system and I/O requirements. The bulk of the manual is devoted to detailed descriptions of the logical 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 Disk and Tape Operating Systems, Assembler Language (Form C24-3414)

IBM System/360 Operating System, Principles of Operation (Form A22-6821)

IBM System/360 Disk Operating System, System Control and System Service Programs (Form C24-5036)

IBM System/360 Disk Operating System, Data Management Concepts (Form C24-3427)

IBM System/360 Disk Operating System, Supervisor and Input/Output Macros (Form C24-5037)



First Edition March 1968

## Figures



## Tables



## Flowcharts

## PURPOSE OF THE ASSEMBLER

The assembler translates a source program coded in IBM System/360 Disk Operating System Assembler Language into a relocatable machine language object program. It performs auxiliary assembler functions designated by the programmer.

Object programs are produced by the assembler in the format required by the linkage editor. The linkage editor produces core image modules which are executable under the control of DOS.

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 AUTOTEST. See Appendix A for details.

The assembler is a language translator, one of the processing programs of the IBM System/360 Disk Operating System. It can operate on IBM System/360 Models 30, 40, 50, 65, or 75 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 system services.

### Main Storage

Main storage requirements include the minimum requirement of the DOS 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.

### Data Sets

The required data sets include one (SYSRES) containing the operating system components, which includes the assembler; three utility data sets (SYS001, SYS002, and SYS003); one input data set (SYSIPT); and a data set (SYSSLB) containing system macro definitions and source coding to be called for through COPY assembler instructions. SYSSLB is kept on SYSRES. There are three optional output data sets as follows:

1. Print data set (SYSLST).
2. Compile and go data set (SYSLNK).
3. Punch data set (SYSPCH).

See Figure 1.

SYSRES. This is a Direct Access Storage Device (DASD) resident data set which contains the control program and other operating system components. The assembler is kept in the Core Image Library on this data set. SYSSLB and SYSLNK are also kept on SYSRES.

SYS001, SYS002, SYS003. These three utility data sets are used as intermediate external storage. Three DASD logical extents, three magnetic tape units, or any combination of the DASD and tape data sets may be used. The use and format of these data sets varies from phase to phase.

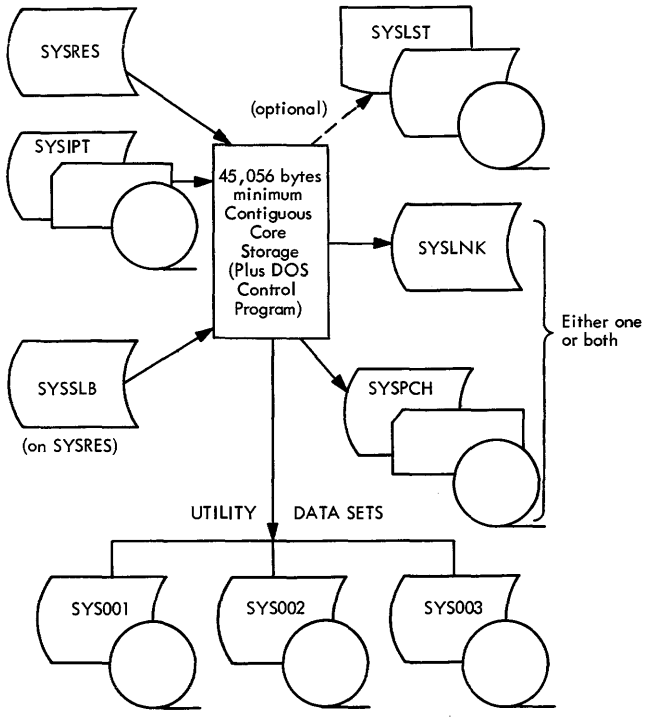



Figure 1. Assembler Data Sets

SYSIPT. This is a DASD, tape, or card reader resident data set containing the source text to be assembled. The source text (also called the text stream) consists of control text, programmer macro definitions, and the main program (also called open code). The format is assumed to be 80-byte record card images, unblocked.

SYSSLB. This is a SYSRES resident 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. It consists of blocked compressed 160-byte records.

SYSLST. This is a printer, DASD, or magnetic tape resident data set containing output text for printing. It consists of 121-byte unblocked print images. The first byte is an ASA control character.

SYSPCH. This data set contains the output text for punching. It may be punch, DASD, or magnetic tape resident. It consists of unblocked 81-byte card images. The first byte is an ASA control character.

SYSLNK. This is a DASD resident data set containing the same output text as SYSPCH, but is used as input for the linkage editor. It consists of blocked 330-byte records with a 2 byte header.

## System Services

Several 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 C, and detailed information is given in the DOS Supervisor and I/O Macros publication.

## THEORY OF OPERATION

The assembler has two major functions: macro generation and assembly. See Figure 2 and Table 1 for a summary of program flow, and program organization.

The generator portion expands macro instructions to one-for-one statements and performs conditional assembly. Phases F0, FCOM, FIN, F1, F2, and F3 comprise the generator portion.

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

There are two other phases. Phase F3E is called if an error condition requires bypassing macro generation. Phase ABT is called if an error condition requires termination of the assembly.

Table 1. Logical Organization of the Assembler

| PHASE | FUNCTION |
|---|---|
| F0 | Master Root Segment - utility data set system I/O macros<br>Macro Generator I/O routines |
| FCOM | Macro Generator Communication area |
| FIN | System I/O macros for SYSIPT and SYSSLB |
| F1 | Program initialization |
| F2 | Initial statement scan<br>System and programmer macro definition editing |
| F3 | Macro instruction generation and merge<br>Conditional assembly |
| F3E | Macro generation error bypass |
| RTA | Assembler control table<br>System I/O macros for SYSLNK and SYSPCH<br>Overflow file I/O routines |
| F7 | Address assignment<br>Symbol processing<br>Literal processing |
| FI | External Symbol Dictionary output |
| F8 | Expression evaluation<br>Program output<br>Relocation dictionary build |
| FPP | Cross-reference, relocation dictionary, and diagnostic processing and output |
| ABT | Assembly error termination |

## Macro Generation and Conditional Assembly Phases

The macro generation and conditional assembly portion of the assembler requires two passes over the source text.

The first pass (in Phase F2) scans the source text, determines the syntax of each statement, and produces edited text records suitable for actual macro generation and conditional assembly.

Variable symbols, sequence symbols, one-for-one statement names, macro instruction names, and symbols appearing in macro instruction operands in the text stream are collected and placed in either a global

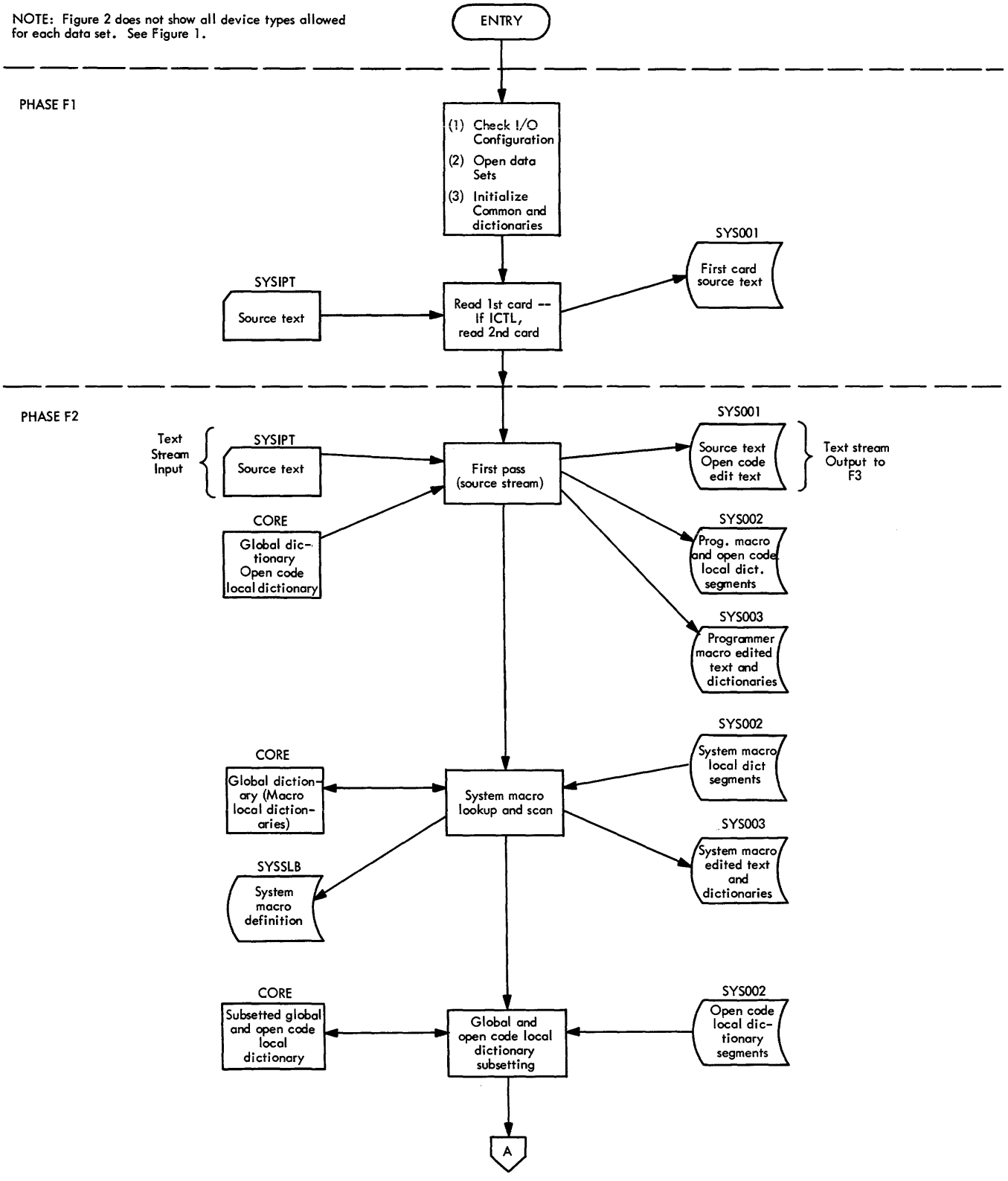



Figure 2. Program and I/O Flow (Part 1 of 3)

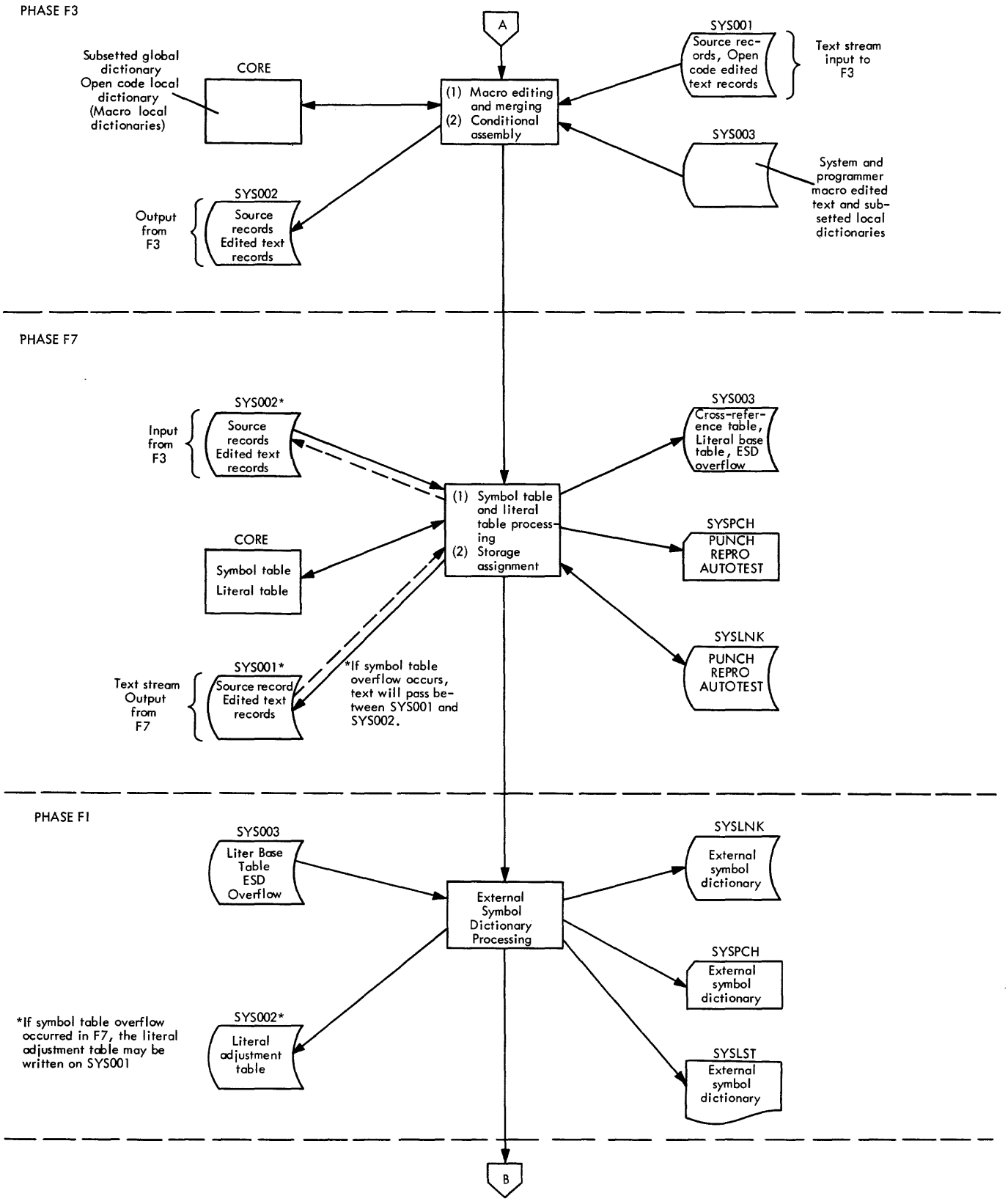



Figure 2. Program and I/O Flow (Part 2 of 3)

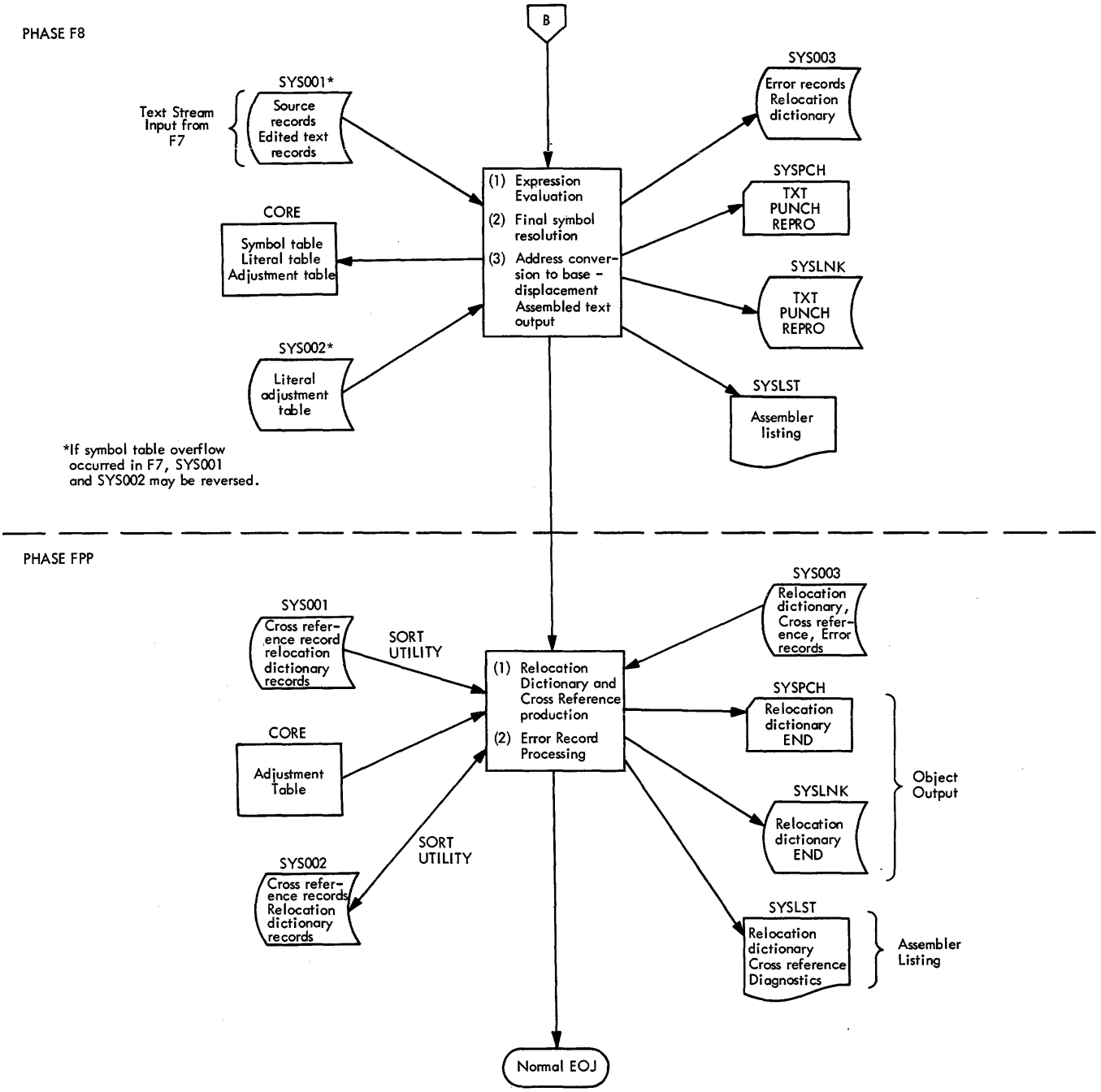



Figure 2. Program and I/O Flow (Part 3 of 3)

dictionary or in one of several local dictionaries, depending on the type of the symbol and the context. Extraneous information is removed from these dictionaries, and subsetted dictionaries are produced for the source text, for each system and programmer macro used in the source text, and for global information. Dictionaries and edited text records are briefly described in Dictionaries and Edited Text which follows. Complete descriptions are in Appendix B.

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

Phase F0 - Master Root Segment

Master Root Segment. This portion of Phase F0 contains the system I/O macros (DTFs and I/O logic modules) for SYS001, SYS002, and SYS003. It also contains linkage to the abort phase in case of EOF or data check on the utility files. The master root segment remains in core for the entire assembly.

Macro Generator I/O routines. This portion of Phase F0 contain READ, WRITE, CHECK, NOTE, POINT, POINTS, POINTR, and POINT W routines for the utility files. They link to the I/O logic modules and the DTFs.

Phase FCOM (Macro Generator Communication Area)

Phase FCOM contains common constants, a translate table, and a communications work area for all macro generator phases.

Phase FIN (SYSIPT and SYSSLB I/O Logic)

Phase FIN contains the I/O logic modules and the DTFs for SYSIPT and SYSSLB.

Phase F1 (Initialization and Assignment)

Phase F1 performs the primary initialization for the macro generation and assembly phases. The operating environment is established by opening data sets for utility, input, and library functions and by initializing dictionaries and common areas.

Phase F2 (Statement Scan)

Phase F2 scans the text stream for programmer macro definitions, the main program, and system macro definitions. It produces edited text for the main program and for each macro definition for input to Phase F3. Phase F2 creates the global and local dictionaries and subsets them for Phase F3. Some syntactic errors are detected and flagged for subsequent error processing. If a macro generation abort condition arises, Phase F2 passes control to Phase F3E, a substitute version of Phase F3.

Phase F3 (Conditional Assembly and Macro Generation)

Using the edited text and dictionaries produced in Phase F2, Phase F3 generates one-for-one edited text statements 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 RTA - Root Segment A

RTA is the root segment for the assembly phases. It contains the Assembler Control Table (a common communications area), the DTFs for SYSLNK and SYSPCH, and the IOCS logic module for SYSLNK, SYSPCH, and SYSLST. RTA also contains a translate table and I/O routines to read and write overflow files.

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 and literal. 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 SYS003 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 SYSLST and SYSPCH and/or SYSLNK, and constructs the literal adjustment table for use in Phase F8. If the external symbol dictionary was written on SYS003 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 SYSPCH and/or SYSLNK and in program listing format on SYSLST.

### 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. Assembly statistics and diagnostic messages are also prepared and written. Phase FPP also writes the END card.

## ASSEMBLER PHYSICAL ORGANIZATION

The assembler is organized as Core Image Library phases and as Relocatable Library load modules. This organization -- including overlay structure, typical load addresses, and external symbols -- is illustrated by the Linkage Editor map in Appendix D. This map also correlates the logical organization of the assembler, described in this manual, with its physical organization. For more information on assembler organization, see IBM System/360 DOS System Generation and Maintenance (Form C24-5033).

### Macro Generator Phases

The F1, E2, and F3 routines link together through address constants located in the Macro Generator Common Area. They use standard DOS/360 linkage conventions. The macro generator phases do not have any overall register usage conventions. Register usage and linkage are described in the comments of each routine listing.

### Assembly Phases

#### Program Levels

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

- Level 0 - Phase initialization, storage allocation, mainline control, phase call, and DOS 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.

#### 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 symbolically reference the Assembler Control Table.

General Register Assignments

General registers 3-15 are referenced symbolically by all assembler subprograms. The equate cards are described in Table 2 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.

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

SRR* (Subroutine Return Address). This general register contains the address through which the current common subroutine 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

All assembly phases routines link through the Assembler Control Table using standard DOS/360 linkage conventions. The exact linkage and the entry and exit points for each routine are in the routine listing comments.

DICTIONARIES AND TABLES

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.

---

*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.

Table 2. General Register Assignments

| Symbol | Tentative EQU | Purpose | Restrictions | Remarks |
|---|---|---|---|---|
| GR0<br>GR1,<br>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.

Hash tables, hashing algorithm, and chaining are the tools employed to accomplish the required data manipulations in the Macro Generator dictionaries and the Symbol Table.

The assembler builds other dictionaries and tables in which the entries are not hashed and chained together. These are the External Symbol Dictionary, the Relocation Dictionary, the Cross Reference Dictionary, the Literal Base Table, and the Literal Adjustment Table. Their entries are stored, read, and written contiguously.

See Appendix D for a complete discussion of dictionary construction methods and Appendix B for a description of dictionary and table formats.

## Macro Generator Dictionaries

The macro generator phases use two types of dictionaries: a global (permanent) dictionary and a local (transient) dictionary. See Figure 3.

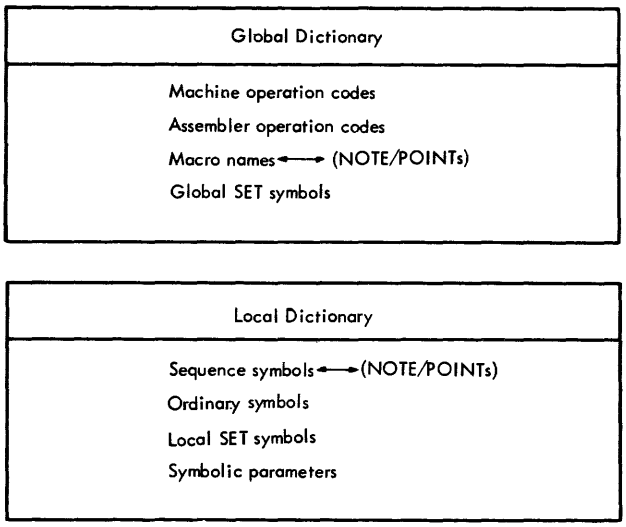



Figure 3. Dictionary Structure

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 macro local dictionary is subsetted at the end of the processing of the macro definition (Phase F2). The subsetted dictionary is written on the SYS002 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 sorts the dictionaries on the "a" pointer and 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

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 in core during Phases F7, FI, and F8. 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 SYS003 if the space is needed by the symbol table.

## INTERMEDIATE TEXT RECORDS

The assembler creates several types of intermediate text records from text stream statements (SYSIPT) and from macro definition and COPY code statements (SYSSLB). These records are of three basic categories:

- Source records
- Edited text (logical) records
- Error records

The record formats are briefly described in the following paragraphs. See Appendix B for a complete description of them.

### Source Record

Source records of statements from SYSIPT are carried from phase to phase without change. They are listed by Phase F8. No source records exist for macro definition statements and COPY code (from SYSSLB) until F8. Then, if the PRINT GEN option is on, they are generated from the edited text records.

### Edited Text Records

Edited text records consist of the source statement fields to which work and code areas have been appended to map and describe the attributes of the fields. These records are created during the initial scan of the source text (in Phase F2). The edited text records are modified during Phase F7 and are converted to object code by Phase F8. During processing, the edited text record for each statement is written on the utility data sets immediately following the source record for that statement.

## Error Records

Errors are detected by Phases F2, F3, F7, and F8. A record for each error is written following the edited text record of the statement for which the error is detected. Up to 16 errors can be detected for each statement. Error records are processed by Phase FPP.

## PHASE F1 - INITIALIZATION AND ASSIGNMENT

### OVERALL OPERATION (FLOWCHART 2)

Phase F1 initializes the assembler. F1 tests for a valid I/O configuration. If the configuration is not valid, it fetches the abort phase. It allocates core for the dictionaries and 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. The first five data sets, SYSIPT, SYSSLB, SYS001, SYS002, and SYS003, are opened. The assignments of the three utility data sets within the assembler are determined by means of the ASSGN commands and job control statements. (See IBM System/360 DOS, System Control and System Services Programs).

Phase F1 reads the first card and, if it is not an ICTL card, branches to Phase F2. If the first card is an ICTL card, it is processed. Another card is read, and F1 then branches to F2.

## OVERALL OPERATION (FLOWCHARTS 3-7)

This phase reads the input from SYSIPT and performs a syntactical scan of all the input. See Figure 4. Any undefined operation codes are assumed to be system macro instructions. A search of SYSSLB 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 performs the following three logical functions:

- Programmer macro definition scan and dictionary build.
- Open code (main) text scan and dictionary build.
- System macro definition scan and dictionary build.

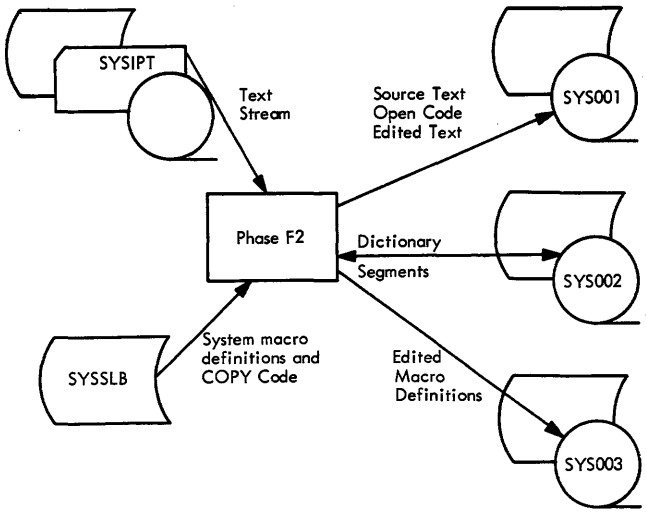



Figure 4. I/O Flow for Phase F2

### Text Stream Scan and Programmer Macro Editing

Phase F2 reads source text from SYSIPT or from SYSSLB (if COPY instructions are encountered). Programmer macro definitions must precede all source statements except ICTL, ISEQ, and listing control instructions.

F2 translates each source statement character from EBCDIC to an internal code (Appendix F). This internal code establishes a continuous collating sequence -- to simplify syntactic scans -- of all valid assembler language characters. Self-defining character strings and comments are not limited to valid assembler characters. Therefore, self-defining character strings are re-translated into EBCDIC when they need to be evaluated.

After translating the statement, F2 writes a copy if it is on SYS001 (See Macro Generator Record Formats in Appendix B.) This copy will be retranslated to EBCDIC by Phase F8 and put out on SYSLST.

F2 now builds edited text from the source statement. (See Macro Generator Record Formats in Appendix B.)

F2 scans the statement again. If the statement has a name field, the name and its attributes are entered in the open code local dictionary or -- if the statement is part of a programmer macro definition -- in the local dictionary being built for that macro.

F2 scans the operation field. If the statement is a prototype, F2 enters the operation code--which is the name of the programmer macro defintion--in the global dictionary. If the statement is not a prototype, F2 looks up the operation code in the global dictionary. If the operation code is not in the dictionary, the statement is assumed to be a system macro instruction (located on SYSSLB) name and is entered in the global dictionary.

F2 scans the operand field. If the statement is part of a programmer macro definition, symbols in the operand field (and their attributes) are placed in the macro local dictionary. For open code one-for-one statements, only sequence symbols or local variable symbols in the operand field are placed in the open code local dictionary -- ordinary symbols are not. For open code macro instructions, all symbols in the operand field are placed in the open code local dictionary.

F2 also checks statement syntax as it scans and it prepares an error record for each error encountered.

The edited text record for open code statements is written on SYS001 immediately following the source record. If the statement is part of a programmer macro definition, the edited text record is written on SYS003. Error records are written after the edited text on both SYS001 and SYS003.

After each complete programmer macro definition is processed, its macro dictionary is subsetted -- the entries are sorted on the little "a" pointer and the symbol, big "A" and little "a" pointers are removed. The dictionary is written on SYS003 following the macro definition and the SYS003 NOTE/POINT location of the dictionary is placed in the global dictionary entry of that macro name. The dictionary in turn points to the programmer macro definition edited text.

If, during source statement processing, a macro local dictionary or the open code local dictionary cannot be contained in core, segments of them are written on SYS002. These segments are backward chained.

## System Macro Editing

After is has processed the text stream, F2 looks up each system macro instruction name entered in the global dictionary. The macro definition is located on SYSSLB, read in, edited, and written on SYS003 in the same manner as each programmer macro definition was processed from the text stream. A macro local dictionary is created during editing and is subsetted and written on SYS003 following the macro definition.

F2 places the SYS003 NOTE/POINT location of each system macro definition local dictionary in the global dictionary entry for that macro instruction name. The dictionary in turn points to the macro definition edited text.

Any name in the global dictionary for which F2 cannot find a corresponding system macro definition on SYSSLB is flagged as an invalid operation code. An error record is written for the statement in which the name appears.

After it has looked up all macro names in the global dictionary, F2 subsets the global dictionary -- removes all entries except those for global variable symbols. F2 also subsets the main local dictionary -- sorts the entries on the little "a" pointer and removes the symbol, big "A" pointers, and little "a" pointers.

The subsetted global and open code local dictionaries remain in core. F2 then FETCHes phase F3.

## 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 is expected, the name (if not already present) and SYS003 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 (GBLA, LCLA, etc.), 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 little "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 little "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 with an indication that the next statement is expected to be a prototype.

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 are scanned and indicated as defined for the dictionary. Control goes to ENDOPR.

START, 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 DCLOSE.

END

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

DCLOSE

DCLOSE is entered after scan of each programmer macro, each system macro, and open code. It closes each macro local dictionary, the open code local dictionary, and the global dictionary after all text has been scanned and all macros edited.

At entry, DRCTY (in macro generator common) contains the note/point address (on SYS002) of the last or only segment of the current local dictionary. DCLOSE first notes the position of SYS002; then it points to the next available position, writes the final block of the current dictionary, and notes its position. DCLOSE then, one by one, reads back the segments of the current dictionary from SYS002 and subsets them, building a subsetted dictionary in core. Subsetted macro local dictionaries are written out on SYS003 immediately following the edited text for their respective macros. If the subsetted dictionary is a programmer macro local dictionary, control returns to DRIVER.

Otherwise, DCLOSE sets a switch in GETSRC to indicate all future input will be from SYSSLB. It then issues a FIND to SYSSLB for each undefined opcode. If the opcodes can be found in SYSSLB, control goes to DRIVER. When no more opcodes can be found in SYSSLB, those remaining are flagged as undefined. The open code local dictionary and the global dictionary are then subsetted, the input stream (SYSIPT) is flushed to /* or end-of-file, and the next phase is FETCHed.

ENDOPR

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

NDSMT3

If in a macro, this subroutine writes on

SYS003; otherwise, it writes on SYS001. 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 SYS003, it is also written on SYS001. 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 SYSIPT or SYSSLB, depending upon which was selected. The record is translated to internal code and immediately written on SYS001 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 data set. In either case, control returns to the caller.

BWFORC

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

BWNOTE

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

## OVERALL OPERATION (FLOWCHARTS 8-11)

Phase F3 reads text from SYS001. See Figure 5. This text includes source, error, and edited records. Source and error records are immediately written onto SYS002. MEND and MEXIT statements are not processed. The edited text for open code and for all macro definitions, plus the subsetted global and local dictionaries, is 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 SYS002. When the end of text is reached, F3 FETCHes Phase F7.

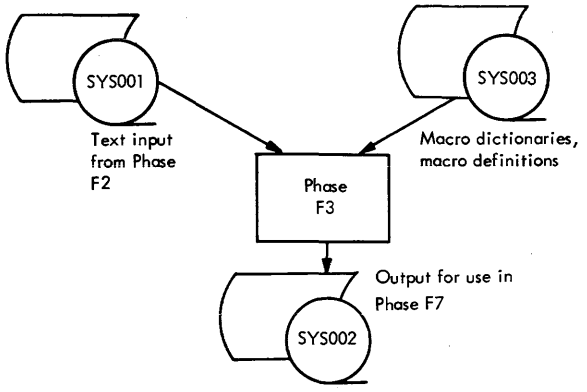



Figure 5. I/O Flow for Phase F3

## PHASE F3E - ABORT CONDITION (FLOWCHART 12)

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

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

All text is read from SYS001. Only source records with accompanying error records are written on SYS002. 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 SYS002, and F3 FETCHes Phase F7.

## FUNCTIONS

Phase F3:

- Generates a parameter table from a macro instruction-macro prototype pair of statements.
- Generates assembler edited text records using macro definition edited text and the information in the global and associated local dictionaries.
- Evaluates conditional assembly expressions.
- Performs conditional assembly.

### Macro Generation

F3 reads text from SYS001. When a macro instruction is encountered, its exact location is NOTEd. The input file is SYS001 for outer macro instructions and SYS003 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 SYS002. When the end-of-macro instruction field is encountered, the position is NOTEd so that the input of text can later be resumed at the correct position. The input data set is then repositioned again to the beginning of the macro instruction text.

The little "a" pointer of macro instruction points to the associated entry in the global dictionary. This entry points to the associated subsetted macro local dictionary on SYS003. If the entry is zero, this mnemonic represents an undefined operation to the macro generator.

If the global dictionary entry is not zero, the first segment of the subsetted local dictionary is read in. This segment contains the macro local dictionary header record which indicates the size of this local dictionary. If there is room in the block of main storage allocated 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 prototype 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 made sequentially in the parameter table. As each prototype 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 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 one-for-one 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 storage 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 storage, further (deeper) nesting of macro instructions is discontinued, and the position of SYS003 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 SYS002, 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 edited text is continued. After completely processing the edited text input from SYS001, F3 transfers control to Phase F7.

### Conditional Assembly

Phase F3 evaluates conditional assembly expressions. If the expression is 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 edited text named by the sequence symbol is obtained from the appropriate local dictionary. The text on SYS001 or SYS003 is repositioned and the appropriate text is read in and processed.

## FUNCTIONAL PROGRAM SECTIONS AND ROUTINES

### ZACINA

This routine is the F3 entry point. It relocates permanent and open code dictionaries, initialized 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

This routine repositions the text file to the sequence symbol.

MENDST

If exit is from an outer macro, this routine sets the text file to read from SYS001; otherwise it continues reading from SYS003. 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 dumps 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 sets the mode switch to character expression. It initializes the pointers, operator table, and result list and it zeros-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 operates on the flags shown in Appendix B.

SYMBL

This is an input pointer 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 it is the end of the expression, TSTOP1 returns to DRIVER.

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.

DOOPR1

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:

- Processing symbols
- Processing literals
- 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 storage allocated 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 encountered are checked for duplicate names, and 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 repositioned to the beginning, and operand processing continues until the previously noted overflow position is reached. At this point, the old table is discarded and processing reverts to assignment mode and the next symbol table segment is constructed.

Building symbol tables and processing operands continue until the last symbol defined has been placed in the symbol table.

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

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

Phase F7 also creates an AUTOTEST symbol table if requested by OPTION SYM, and writes cards generated by PUNCH and REPRO statements.

Source, error, and edited text is read from SYS002, and literal pools and text are written on SYS001.

Overflow of the external symbol dictionary table, the literal-pool base table, or cross-reference table will result in writing overflow segments on SYS003.

## I/O FUNCTIONS

Phase F7 may make several passes at the source text. See Figure 6. On the initial iteration, the text is read from SYS002 and reblocked onto SYS001. On subsequent iterations, the text is passed between SYS001 and SYS002.

The text is reformatted so that a "broken" record always starts at the beginning of a physical block. This ensures that a logical record is 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 output to a utility file.

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

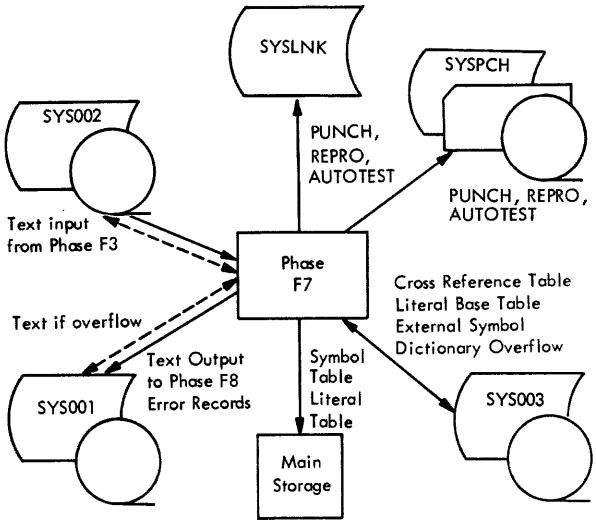



Figure 6. I/O Flow for Phase F7

As the cross-reference block and literal base table overflow, they are written onto the overflow file, SYS003. Each time the external symbol dictionary overflows it is written onto SYS003, 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.

PHASE ORGANIZATION

Phase F7 is organized as follows:

- Mainline control - F7C.
- Functional routines.

  GET statement - F7X.
  DC, DS evaluation - F7D.
  External symbol dictionary processor - F7E.
  AUTOTEST processor - F7N.

- Common subroutines (common to routines within Phase F7).

  Symbol table function - F7S.
  Expression evaluation - F7V.
  Error logging function - F7L.
  Literal DC generator - F7G.
  I/O subroutines and initialization - F7I.

F7C - Mainline Control (Flowcharts 13-18)

Control is passed to F7C by the initialization routine, F7I.

Program modifications are made based on the setting of the cross-reference and AUTOTEST option bits. F7N is called by F7C to generate AUTOTEST cards for all edited statements if the option bit is set.

F7X 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 F7D 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, F7C tests all records for symbol references. If substitution is required, a work bucket is attached for each symbol referenced in the operand.

For machine operation codes, F7C tests name fields for multiply defined symbols and evaluates literals for duplicates.

For assembler operation codes, F7C processes assembler operations in the substitution mode only when substitution is required.

F7X - Phase F7 GET Statement Routine (Flowchart 19)

F7X is used by 7C 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, F7X puts the current text record onto the text file by calling PUTXT.

If an error record is in the build area, F7X 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, F7X calls GETPT for a pointer to the next text record in the input buffer.

If an end-of-file is detected, F7X moves a QUIT record into the text work area if the mode is assignment. If the mode is substitution, the file is repositioned to the first text record.

F7X tests each edited text record for a record type. If the record is not edited text, 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, and substituted fields are adjusted for leading and trailing blanks.

F7X sets absolute pointers to the operand field and appends fixed fields and symbol work buckets (if any exist).

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.

## F7D - DC/DS Evaluation Routine (Flowchart 20)

This routine is called by F7C to process DCs, DSs, 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, and literal DC operand for use by the Phase F8 DC evaluation routine (F8D). In the external symbol dictionary, a table entry is made for each valid constant in a V-type DC.

One complete statement is processed in each pass, and control is returned to F7C.

## F7E - External Symbol Dictionary Processor (Flowcharts 21-23)

This processor is called whenever any of the assembler operations COM, START, CSECT, DSECT, ENTRY, EXTRN, or ORG or a V-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.(See Appendix G.)
- Issuing various error messages.

## F7N - F7 AUTOTEST Routine

If the AUTOTEST option is exercised, control is passed from F7C to F7N after the process of each statement that defines a symbol or affects the location counter in any way.

The output records (cards) of F7N are written on SYSPCH and/or SYSLNK data sets. Subsequent executions of the object program in the AUTOTEST mode use this information.

## F7S - Symbol Table Subroutine (Flowchart 24)

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.

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 overflow can occur and room can be made for the symbol table, or if the ESD processor made the call, the external symbol dictionary is put on the overflow file. If not, the need to enter substitution mode is signalled.

## F7V - 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 F7V to terminate. The terminating character is always a left or right parenthesis, blank, or comma, unless there was a syntactical error, in which case SP1 is a 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 F7V, 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 F7V; errors other than syntax are logged, and normal processing is continued.

F7V has the following functions:

- Evaluate expressions.
- Log type 2 cross-references (XREF) - (only F8V).
- Convert self-defining values (SDVCF).
- Detect the following errors and pass the information to F7L:

  Relocatability error
  Self-defining value too large
  Arithmetic overflow
  Symbol not found - (F8V)
  Symbol not previously defined - (F7V)
  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), a 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 halfword, in effect putting zeros into that position, since all the tables are zeroed during F7V initialization.

Type 2 cross-references are made during Phase F7 assignment mode.

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 intially set to 0 on entry to F7V. 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 3. For example, / is valid only when COND=3 (/ follows an absolute term or left parenthesis.). 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 parenthesis).

Work tables used in F7V 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.

Table 3. 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 IJY026 | |
| ) | 0/1/2 | log error IJY085 | RPAR |
| | 3/4 | set COND = 3 if PCNTR = 0, end of expression | |
| + or - | 3/4 | set COND = 1 | LTCOM |
| | 0/1/2 | log error IJY085 | LOOP |
| * | 0/1 | set COND = 4 (* is location counter ref.) | LTCOM |
| | 2 | log error IJY085 | |
| | 3 | set COND = 2 (* is mult.) | |
| | 4 | log error IJY025 | |
| / | 3 | set COND = 2 | LTCOM |
| | 0/1/2 | log error IJY085 | LOOP |
| | 4 | log error IJY025 | |
| absolute term | 3/4 | log error IJY085 | Term Computing Routines |
| | 0/1/2 | set COND = 3 | |
| relocatable term | 0/1 | set COND = 4 | |
| | 2 | log error IJY025 | |
| | 3/4 | log error IJY085 | |
| , or blank | 0/1/2 | log error IJY039 | BLCOM |
| | 3/4 | if PCNTR > 0, log error IJY039 | |

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 halfwords) - 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 unpaired positive term in the first halfword, and the remaining 15 halfwords will contain zeros. A complex relocatable expression at exit will have a non-zero halfword for each unpaired relative term. The non-zero halfwords 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.

## F7L - Error Logging for Phases F7 and F8 (Flowchart 26)

This routine is called to attach error messages to an edited text record.

F7L tests the error switch in the control table to determine if there is an error record for the current text record 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, F7L 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.

## F7G - Literal DC Generator (Flowchart 27)

F7G is a routine which builds a literal DC

edited text record for an outstanding literal entry in the symbol table. F7X invokes F7G once for each literal DC that is to be built into the edited text. F7G 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.

F7I - Phase F7 Initialization and I/O

The I/O portion of phase initialization OPENs the three utility data sets. READ, WRITE, CHECK, and POINTS are employed for all 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.

F7 I/O functions for PHCLS, GETPT, GETXTM, PUTXT, CLSTXT, CWRESD, CRDESD, PUTXRF, and PUTLBT are described below.

PHCLS - Phase Close

The I/O portion of the phase close function repositions (POINTS) SYS001 and SYS002 and inserts the following parameters into the I/O portion of the assembler control table.

CTRLBT (4 bytes) - Pointer to the first literal base table block on the overflow file, SYS003.

CTCLBT (2 bytes) - Count of the number of literal base table physical blocks which have been written onto SYS003.

CTRXRF (4 bytes) - Pointer to the first cross-reference table block contained on the overflow file, SYS003.

CTCXRF (2 bytes) - Count of the number of cross-reference physical blocks which have been written onto SYS003.

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, SYS003. 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. PHCLS FETCHes FI.

GETPT - Get Point

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

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.

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 continuous 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.

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

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 repositioned. 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, SYS003, and NOTEs its position for future reference.

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.

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.

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.

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.

SYPUNH - System Output (Located in RTA)

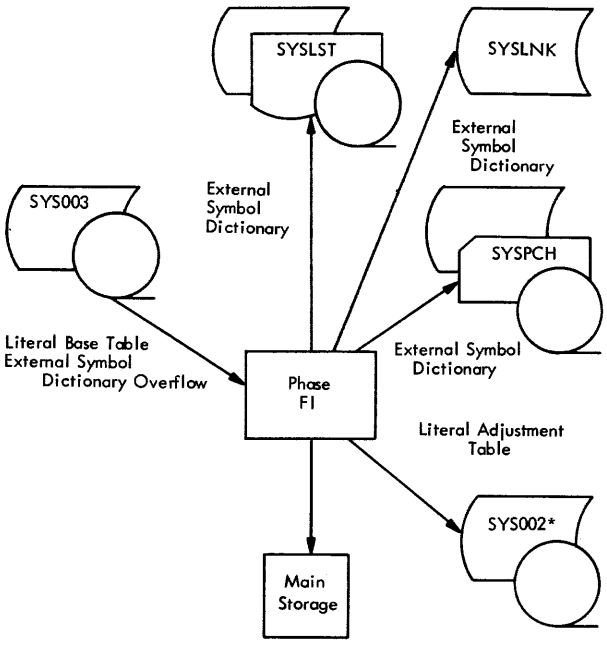
SYPUNH outputs 80-character logical records to either the SYSPCH or SYSLNK data sets or to both. On entry, SP1 points to the first byte of an 81-character buffer where the first character is an internal control character. SYPUNH tests the SYSPCH and the SYSLNK option bits. If either one is set, the contents of the 81-character buffer (except for the control character) are transferred accordingly to the SYSPCH or SYSLNK 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 SYSLST, SYSPCH, and/or SYSLNK data sets. External symbol dictionary segments and literal pool bases are located either in main storage or in an overflow segment on SYS001.

## 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 7. 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 perform their own blocking and deblocking functions for the logical record entries. The system output routines SYPUNH and SYSL are called to output external symbol dictionary data.




*If symbol table overflow occurred in Phase F7, SYS001 may be used instead of SYS002.

Figure 7. I/O Flow for Phase FI

## I/O SUBROUTINES

### FII - FI Initialization

The I/O portion of phase initialization determines if any literal base table entries have been output from Phase F7. If so, the first literal base table block is read from the overflow file, SYS001, and the pointers to the first logical record entry within the block are initialized. Control is then transferred to the mainline control driver.

Phase F7I (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. SYS001 and the alternate overflow file are repositioned, 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.

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.

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, SYS003, 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, SYSLST.

Method. The SYSL option bit is tested. If set, the line is written on SYSLST. If not set, a return is executed.

MAINLINE 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, or XD type.

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

## OVERALL OPERATION (FLOWCHARTS 29-36)

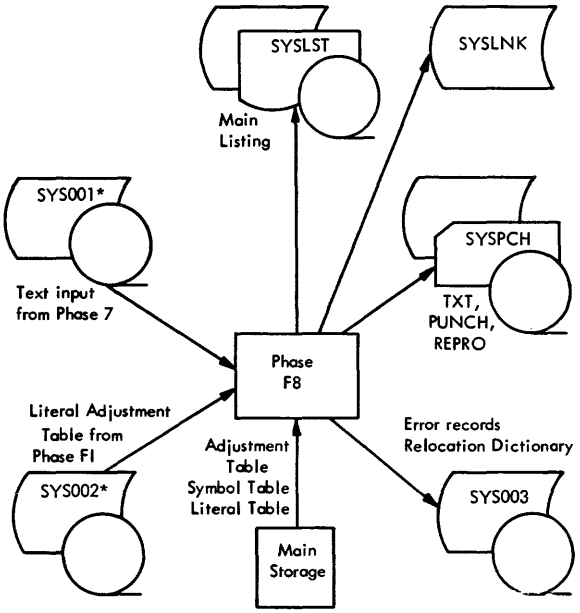
Phase F8 makes the final pass through the program text, which is read from SYS001 or SYS002, depending on the number of iterations. During this pass, any operands which were not processed by Phase F7 are processed from the last symbol table created during that phase.

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 SYSPCH or SYSLNK, and in program listing format on SYSLST.

Invalid statements are flagged on SYSLST. Error records are created and written on the overflow file, SYS003, to be listed by the Post-Processor Phase.

The relocation dictionary is built during Phase F8, and segments can overflow onto SYS003.




*If symbol table overflow occurred in Phase F7, SYS001 and SYS002 may be reversed.

Figure 8. I/O Flow for Phase F8

## I/O FUNCTIONS

Phase F8 passes through the text stream once scanning the logical text records which were output from Phase F7. See Figure 8. The phase inputs the literal adjustment table blocks from the alternate overflow file and outputs cross-reference, relocation dictionary, and diagnostic records onto the overflow file for subsequent processing by FPP. 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. In addition, records are output to the SYSLST, SYSPCH, and SYSLNK files via the output subroutines SYSL and SYPUNH.

## PHASE ORGANIZATION

The following control sections comprise Phase F8: F8I, F8C, F8M, F8A, F8P, F8D, F8V, F8L, F8N, and F8S. These are self-contained routines which may be link-edited separately. Communication between routines is via registers and the ACT table. When FI passes control to Phase F8, the above routines are loaded into core and remain there until the completion of Phase F8 execution.

### F8I - Phase F8 Initialization and I/O

On entry to Phase F8, the overflow file, SYS003, 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. 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, F8C.

## F8I 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, SYS003. The first block written onto the overflow file is NOTEd for future reference.

### WTERR - Write Error Message

WTERR outputs a printer-formatted diagnostic message 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 a 121-byte error message.

### PHCLS - Phase F8 Close

The phase close subroutine POINTS 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 SYSPCH, SYSLNK, LIST, XREF, and TEST.

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 FPP, the Post Processor Phase.

## F8C - Mainline Control (Flowchart 29)

Control is passed to F8C for each record; F8C 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.

## F8C 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 F8I 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.

## F8M - Machine Operation Processor (Flowchart 30)

F8M processes the operand field of all machine ops. Decomposition, adjustment, and formatting occur in this routine. The decomposition routine using table is shown in Figure 9. F8M is entered from the F8C routine.

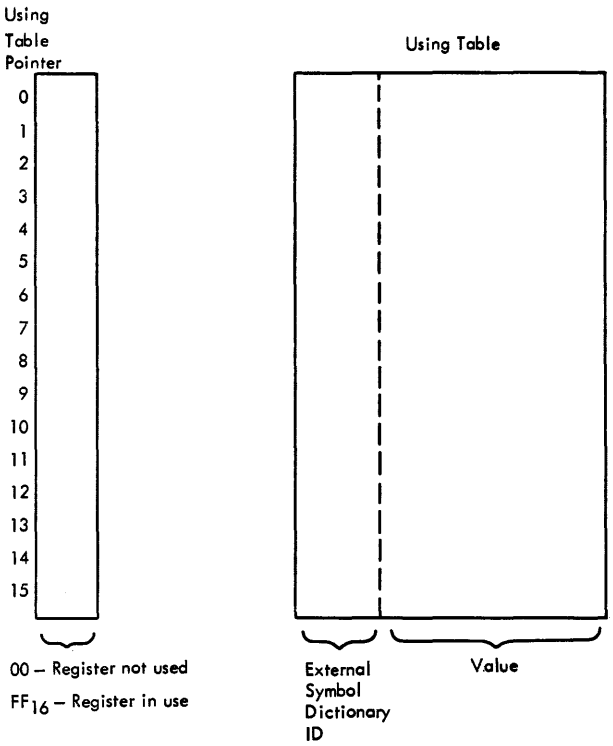
## F8M Subroutines

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 10. 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.




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 9. Decomposition Routine Using Table

## F8A - Assembler Operation Processor (Flowchart 31)

F8A processes the following assembler ops: MNOTE, PRINT, SPACE, EJECT, PUNCH, REPRO, TITLE, ENTRY, EXTRN, START, CSECT, DSECT, COM, EQU, ORG, END, LTORG, USING, DROP, literal DC, DC, DS, CCW, and CNOP.

F8A is entered from the F8C routine.

## F8A 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.

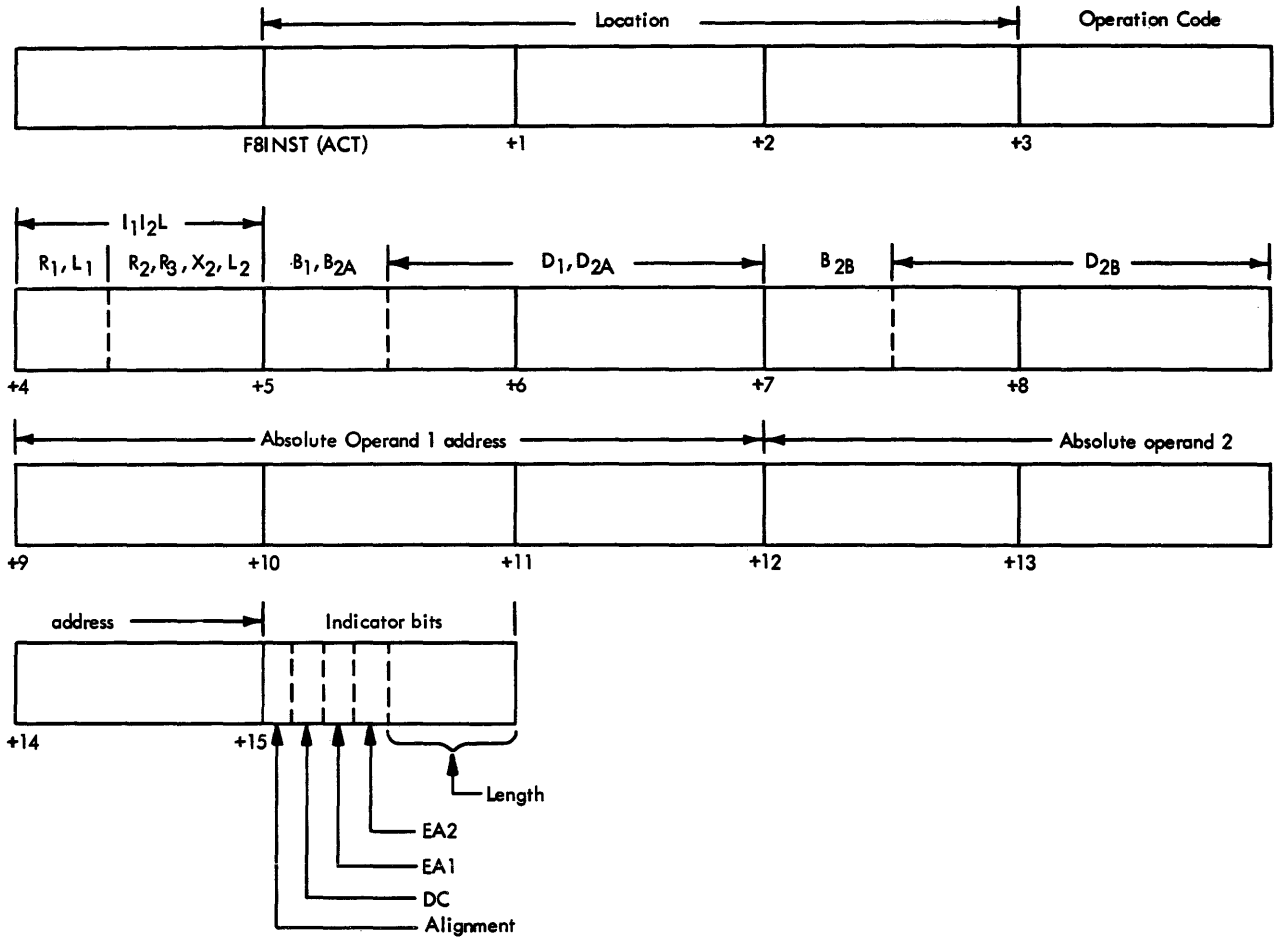



Figure 10. Instruction Building Area

| | | | | |
|---|---|---|---|---|
| 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 is set to the number of lines to be spaced.

EJECT

Sets a switch, EJCSTW(ACT), to $FF_{16}$.

PUNCHB

Sets a switch, REPSW(ACT), to 1.

REPRO

Sets a switch, REPSW(ACT), to 3.

TITLEB

Sets a switch, REPSW(ACT), to 7.

MNOTST

Sets a switch, REPSW(ACT), 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 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 mainline control.

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 F8M, 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 F8D 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 equivalence 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 for each register being used, and the correct value and the external symbol dictionary ID are 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

Switches are set for proper branching within the F8D routine, and control is passed to F8D.

F8P - 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.

F8P has four entry points: F8P, BLDIMG, COMMENT, and LOADRA. Text records are passed to F8P as they are encountered in the text stream.

- F8P processes edited text records (type 100).
- BLDIMG builds a source image from generated edited records (type 110 or 111).
- COMMENT loads the right side for source records (type 000, 010, or 011) and puts an error line for error records (type 001).
- 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):

- F8P 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.

F8P has the following functions (acronyms within parentheses 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.

F8D - DC Evaluation (Flowcharts 34-35)

Phase F8 mainline control calls F8D each time a DC, DS, or literal DC is encountered. The low order byte of register 13, at entry, is set as follows: 00 = DS, FF = DC, mixed = literal DC.

F8D processes an entire DC statement each pass. The print routine, F8P, 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 B5. Exceptions are character, hexadecimal, and binary type DCs which may contain only one constant per operand.

Relocation dictionary entries are made for A-, Y-, and V-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.

Upon completion of a statement, F8D returns to F8C.

## F8N - 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 (F8D) 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 F8D evaluation with three pointers: Register No. 1 points to the right delimiter, Register No. 2 points to the converted value of the constant (eight bytes), and Register No. 13 points to an error flag. If Register No. 13 is zero, no error has occurred.

## F8V - Expression Evaluation Subroutine

This subroutine is the same as F7V with the exception of XREF, but the two are loaded separately in their respective phases.

## F8L - Log Error Subroutine

This subroutine is the same as F7L, but the two are loaded separately in their respective phases.

## F8S - 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, FPP, reads the RLD and cross-reference records from the overflow file (SYS003), sorts them, writes RLD on SYSPCH and/or SYSLNK and SYSLST and the XREF records on SYSLST. FPP also writes the END object card on SYSPCH and/or SYSLNK after the RLD. See Figure 11. In the event there are too many cross-reference record entries to be held in main storage at one time, SYS001 and SYS002 are used for sorting.

The second section, FD, reads error records from the overflow file (SYS003) and formats an error message by matching a code in the error record against a message table in main storage. FD also prints the statistical information (statements flagged), and returns to the DOS control program.

## FPP FUNCTIONS (FLOWCHART 37)

FPP produces the relocation dictionary and the cross-reference list, if requested.

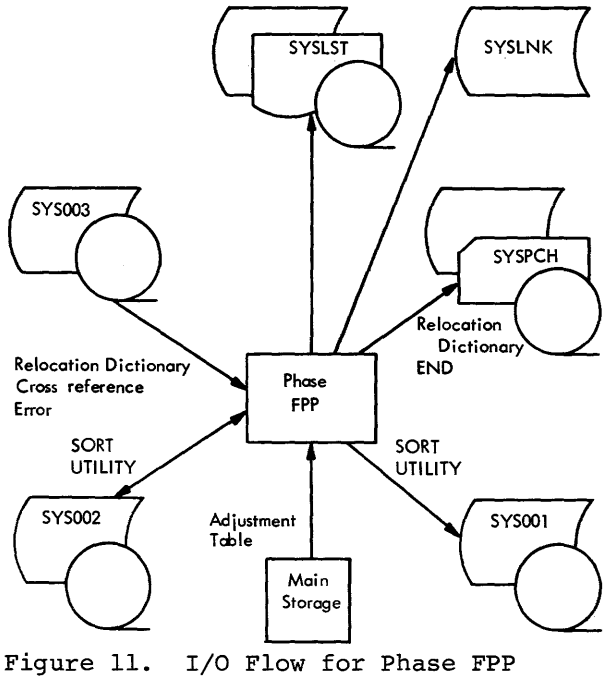



Figure 11. I/O Flow for Phase FPP

FPP sorts the relocation dictionary entries by address. The sorted dictionary is written onto SYSLST and SYSPCH and/or SYSLNK. The loader END record is constructed and written onto SYSPCH.

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

FPP branches to Section FD.

## FPP SUBROUTINES

### GTOX - Cross-Reference List Checker

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

### 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 SYSLST, SYSPCH, and/or SYSLNK.

### SETOT2 - Cross Reference Writer

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

### WR1XRF - Cross-Reference Writer

This routine writes cross-reference on SYS001 (or SYS002) for sorting.

### XRFLOD - Cross-Reference Starter

This routine starts the cross-reference input pass.

Other FPP Subroutines

- CHKSWH - Jump Table Sort
- FPP - Phase Initialization (FPP)
- EPRLZ - Merge Tape Writer
- EP2 - Merge Tape Writer
- ESORT - Input Data Sorter
- GTOR - Control Table Reader
- PPIN - Phase PP Initialization
- RDlXRF - Cross-Reference Record Merger

FD FUNCTIONS (FLOWCHART 38)

The Phase FPP diagnostic, FD, reads the error records from the overflow file (SYS003). 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 SYSLST.

The functions of this phase are as follows:

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

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 it 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, and the phase returns to the caller. 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. The phase exits to the DOS control program.

FD 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.

GETERR - Error Record Reader

This routine reads error records.

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.

ML10 - Statement Printer

This routine prints the total number of statements flagged.

ML11 - Error Message Lister

This routine lists the error messages.

This phase is entered when it is necessary to abort an assembly. An appropriate message is put out to SYSLST and SYSLOG and the run is terminated via EOJ.

Two categories of problems force an abort: uncorrectable I/O errors and improper environment. The first entry is via the ERROPT and EOFADDR exits of the DTFs. It is recognized by the fact that General Register 15 contains a value (the DTF address). In this case the offending unit is recognized by the unit code contained in the DTF. If this code is unrecognizable or is the code for SYSLST, the message is put out on SYSLOG only.

If General Register 15 is zero on entry to the abort phase the problem is assumed to be improper environment. In all cases the run is terminated by EOJ and the remainder of the job is bypassed.

General Registers 2 and 3 are assumed to contain up to 8 error codes of 1 byte each. A message is prepared for each code.

## FLOWCHARTS

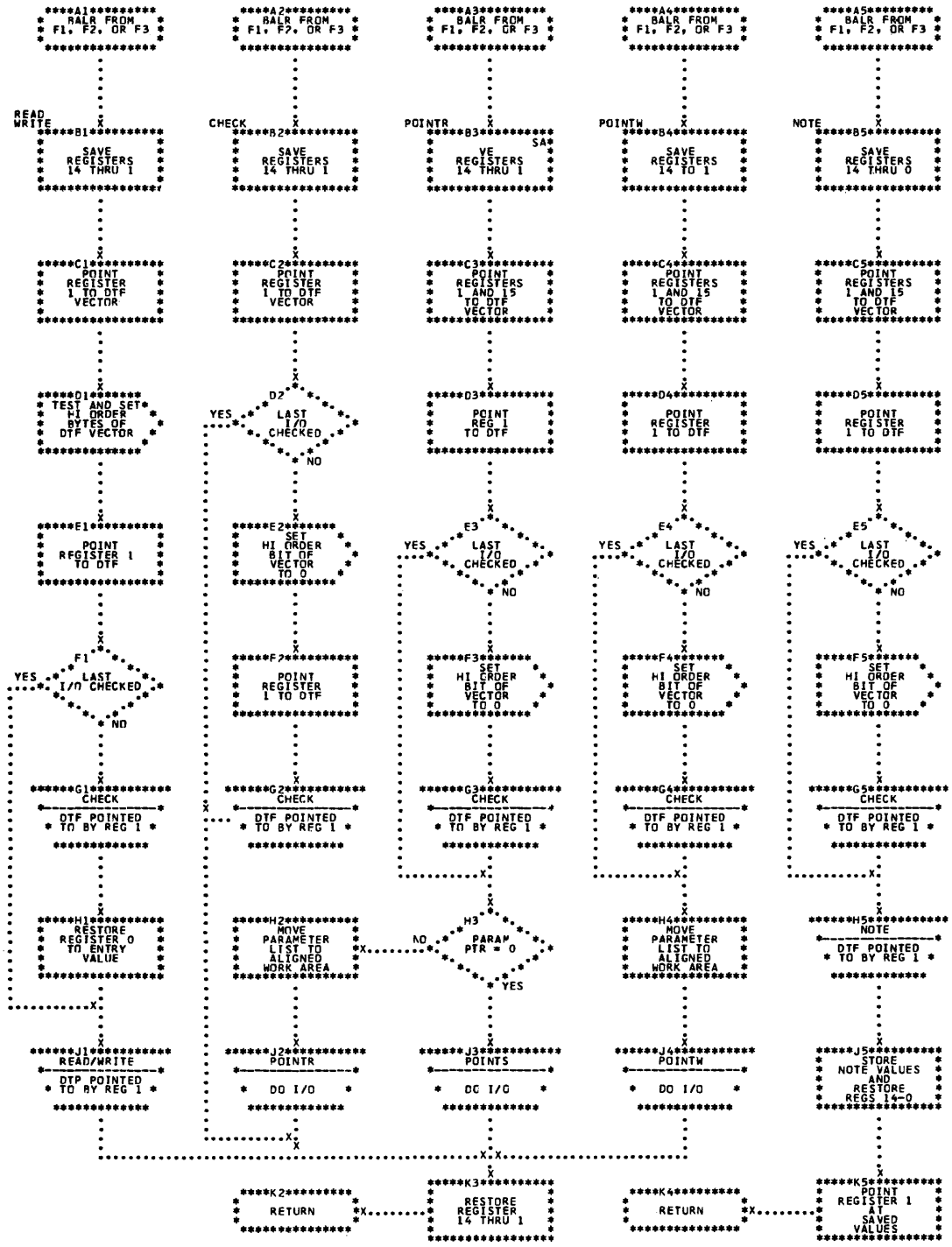



Chart 1. MAC - Macro Generator I/O

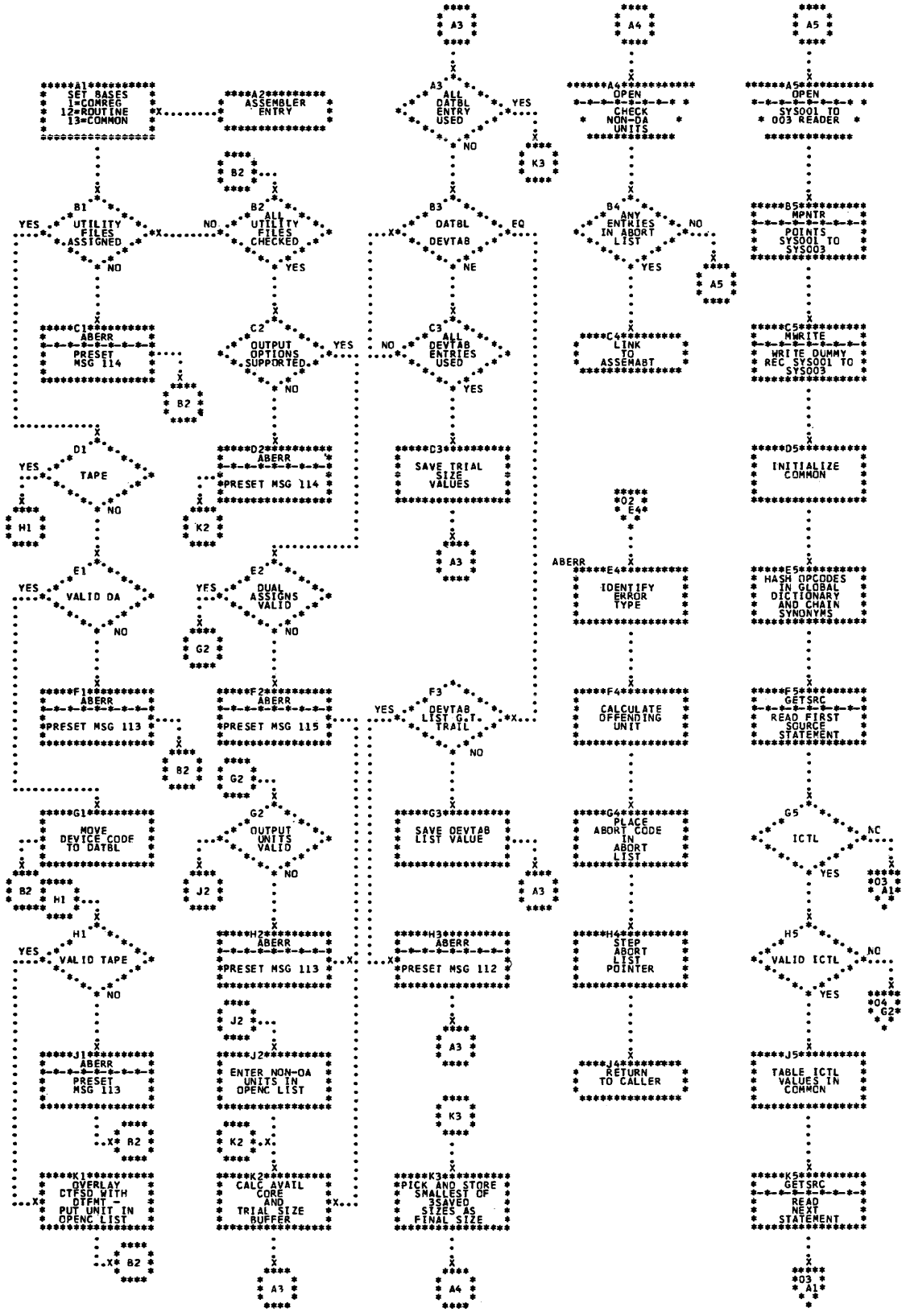



Chart 2. F1 - Phase F1

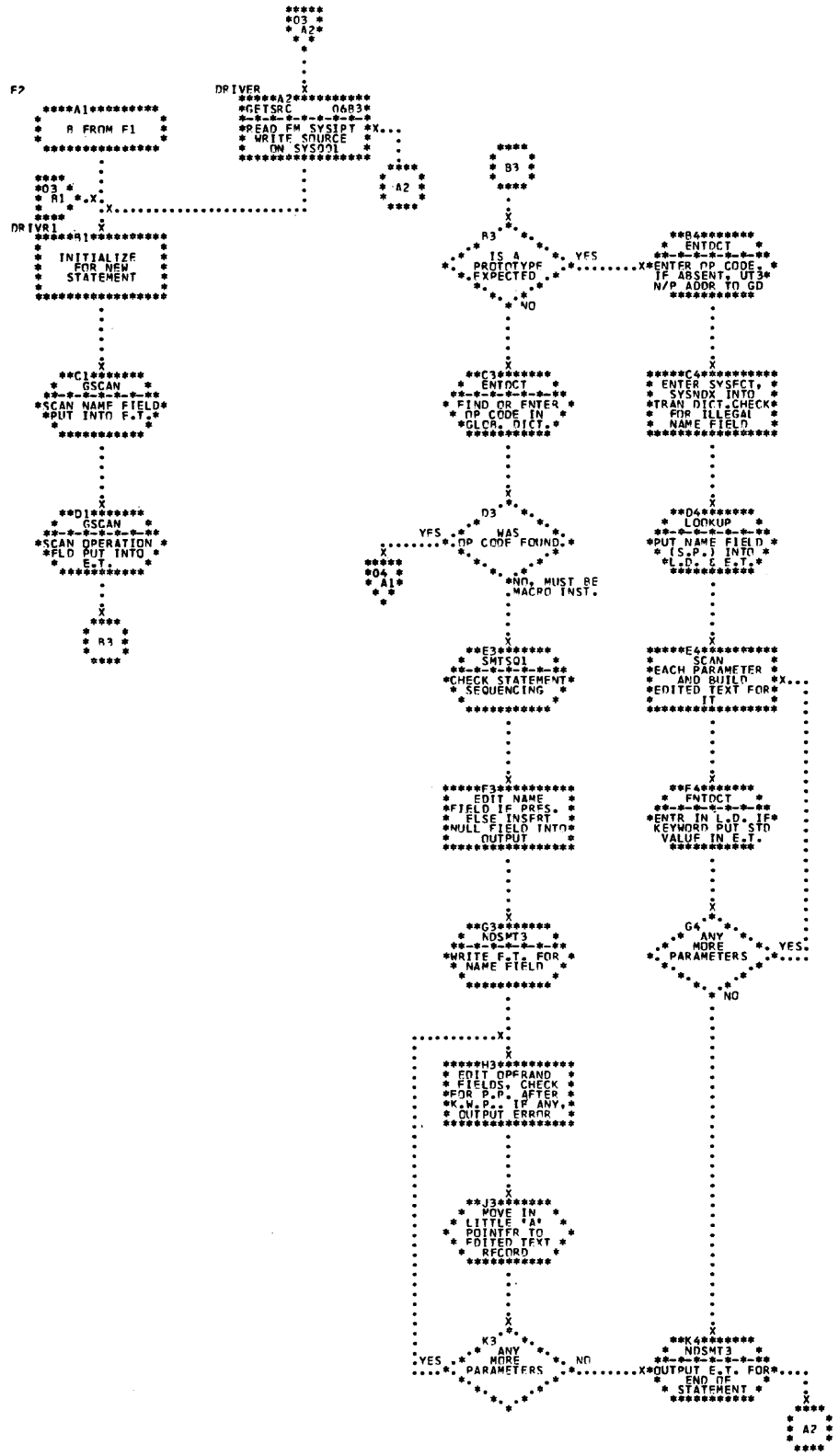



Chart 3. F2 - Phase F2 (1 of 5)

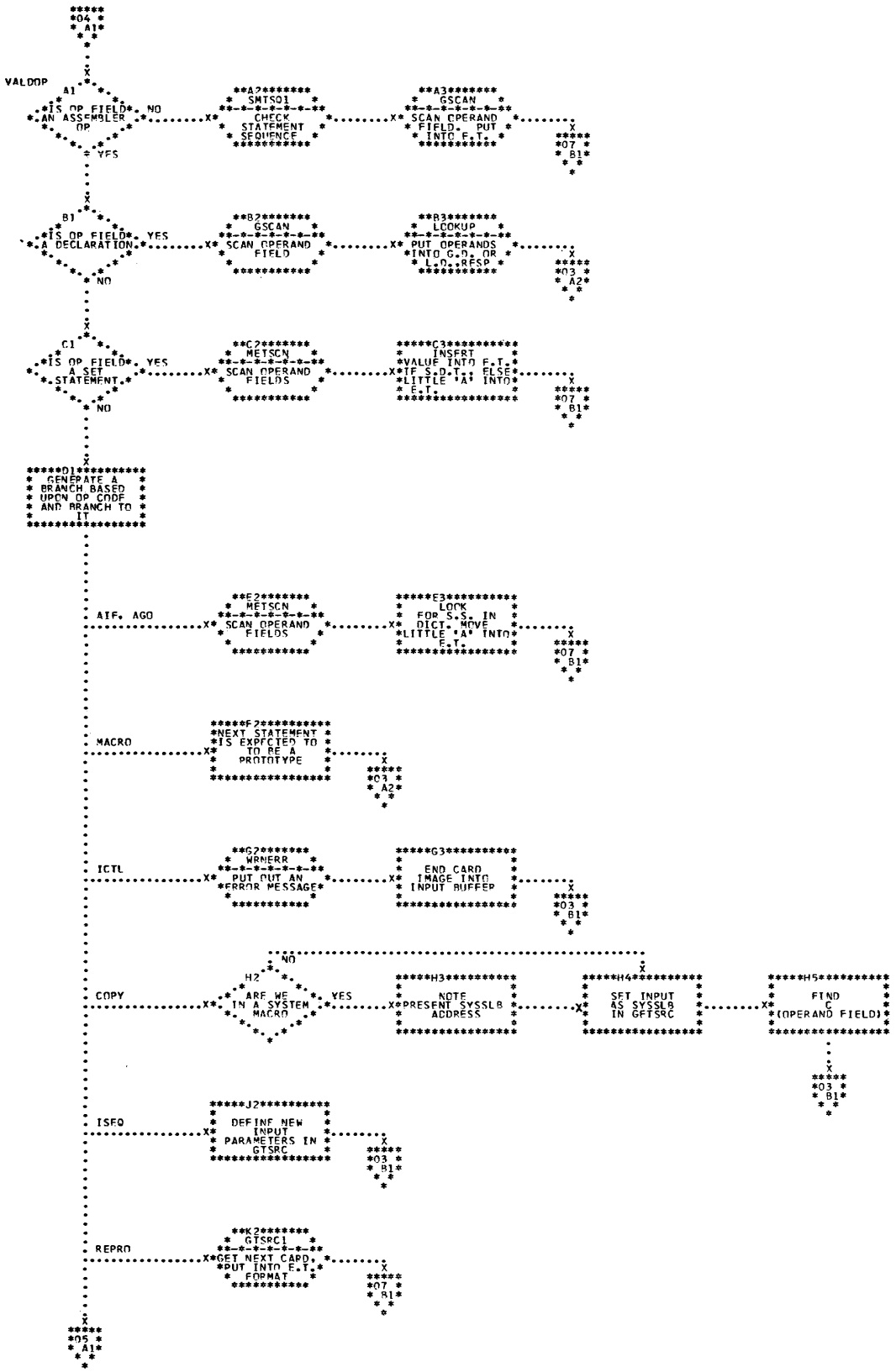



Chart 4. F2 - Phase F2 (2 of 5)

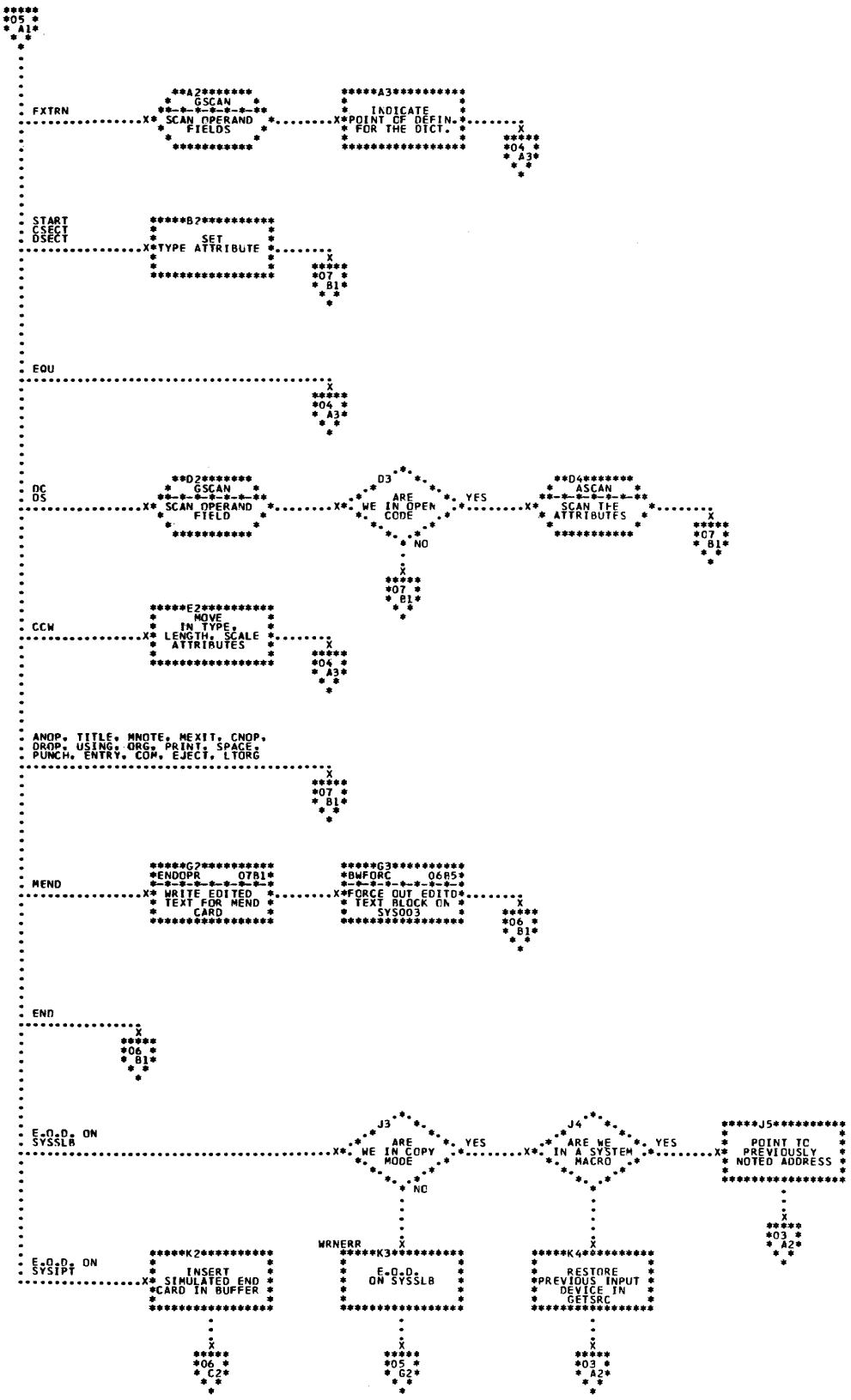



Chart 5. F2 - Phase F2 (3 of 5)

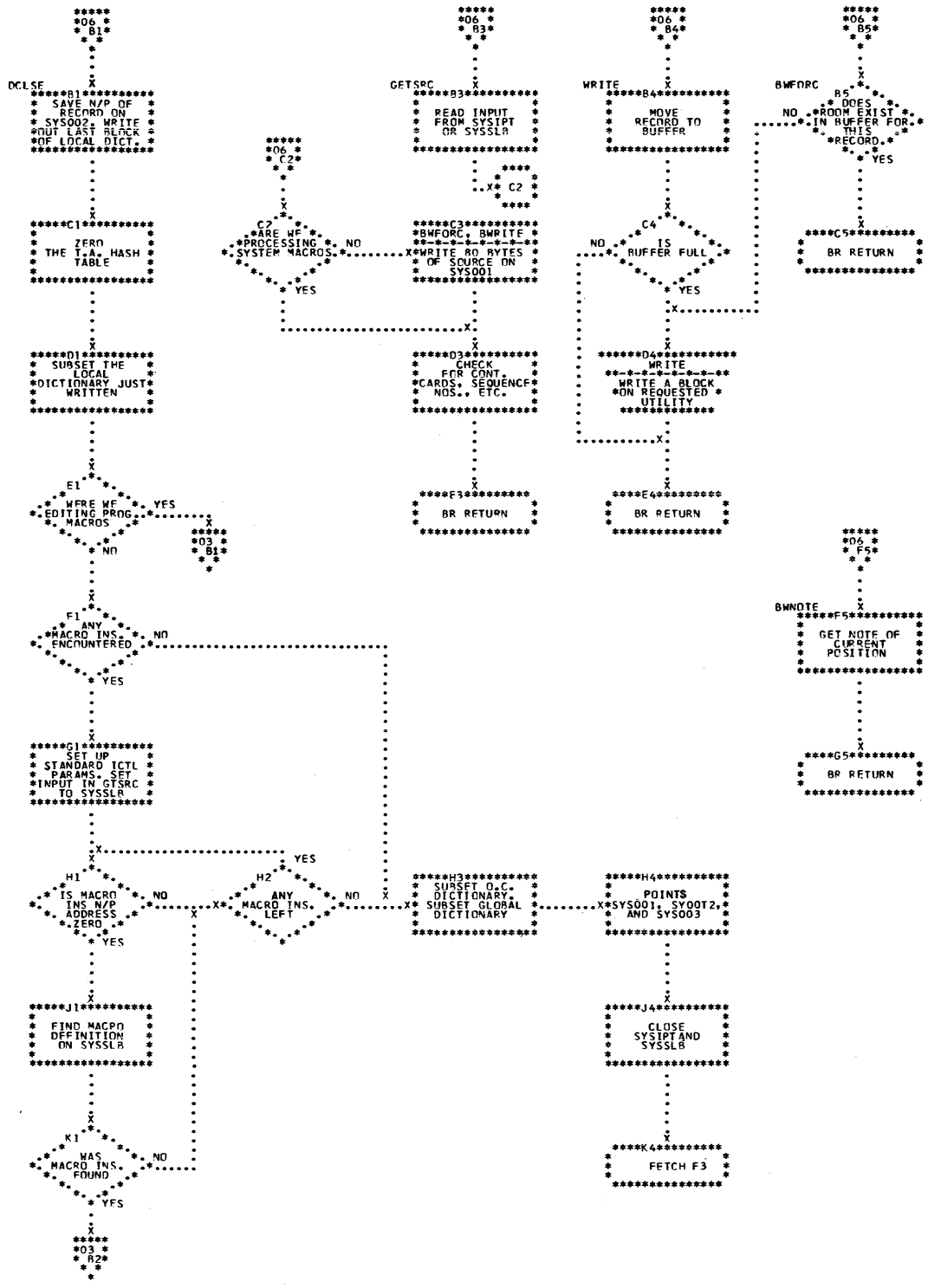



Chart 6. F2 - Phase F2 (4 of 5)

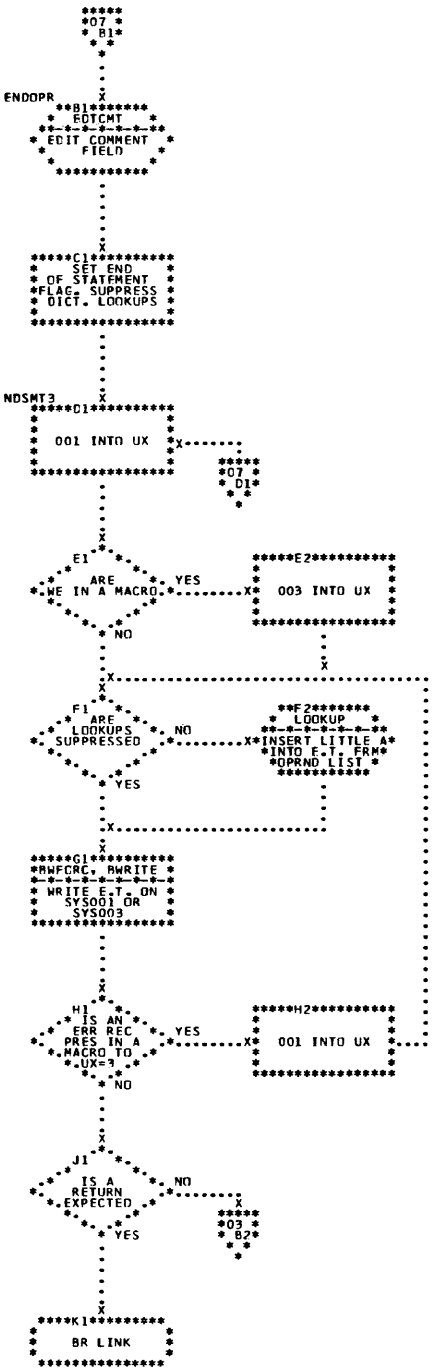



Chart 7. F2 - Phase F2 (5 of 5)

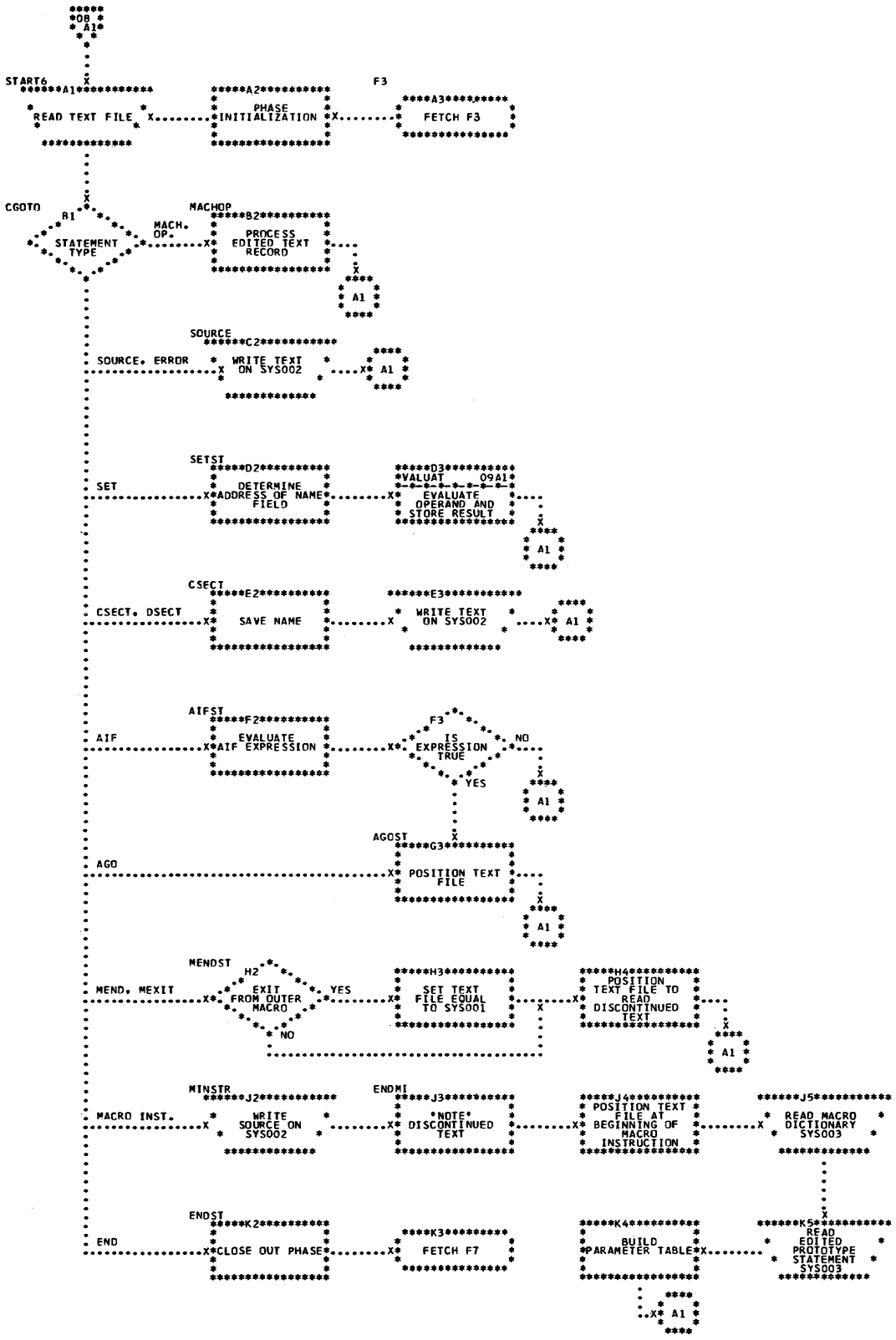



Chart 8. F3 - Phase F3 Mainline Control

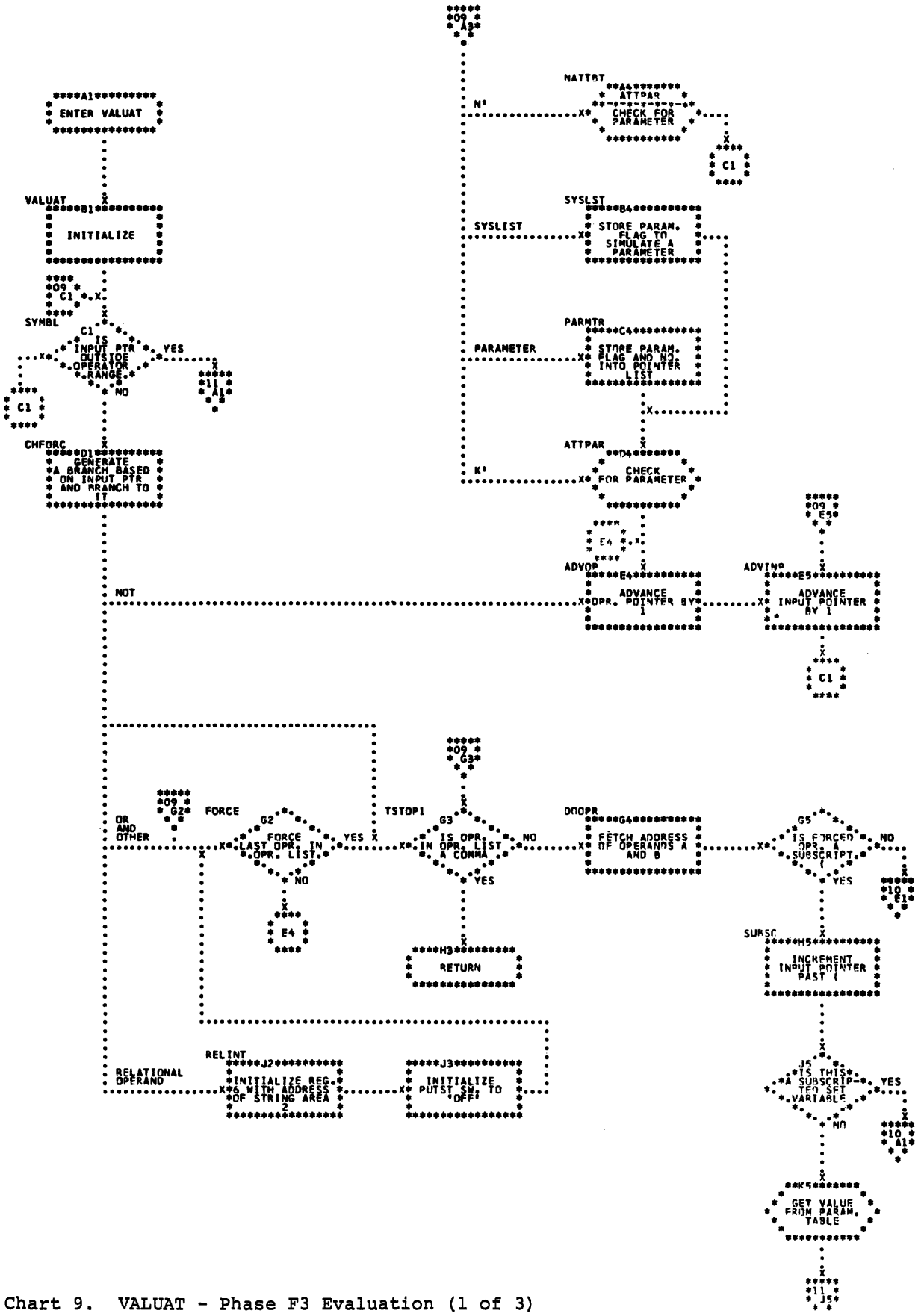



Chart 9. VALUAT - Phase F3 Evaluation (1 of 3)

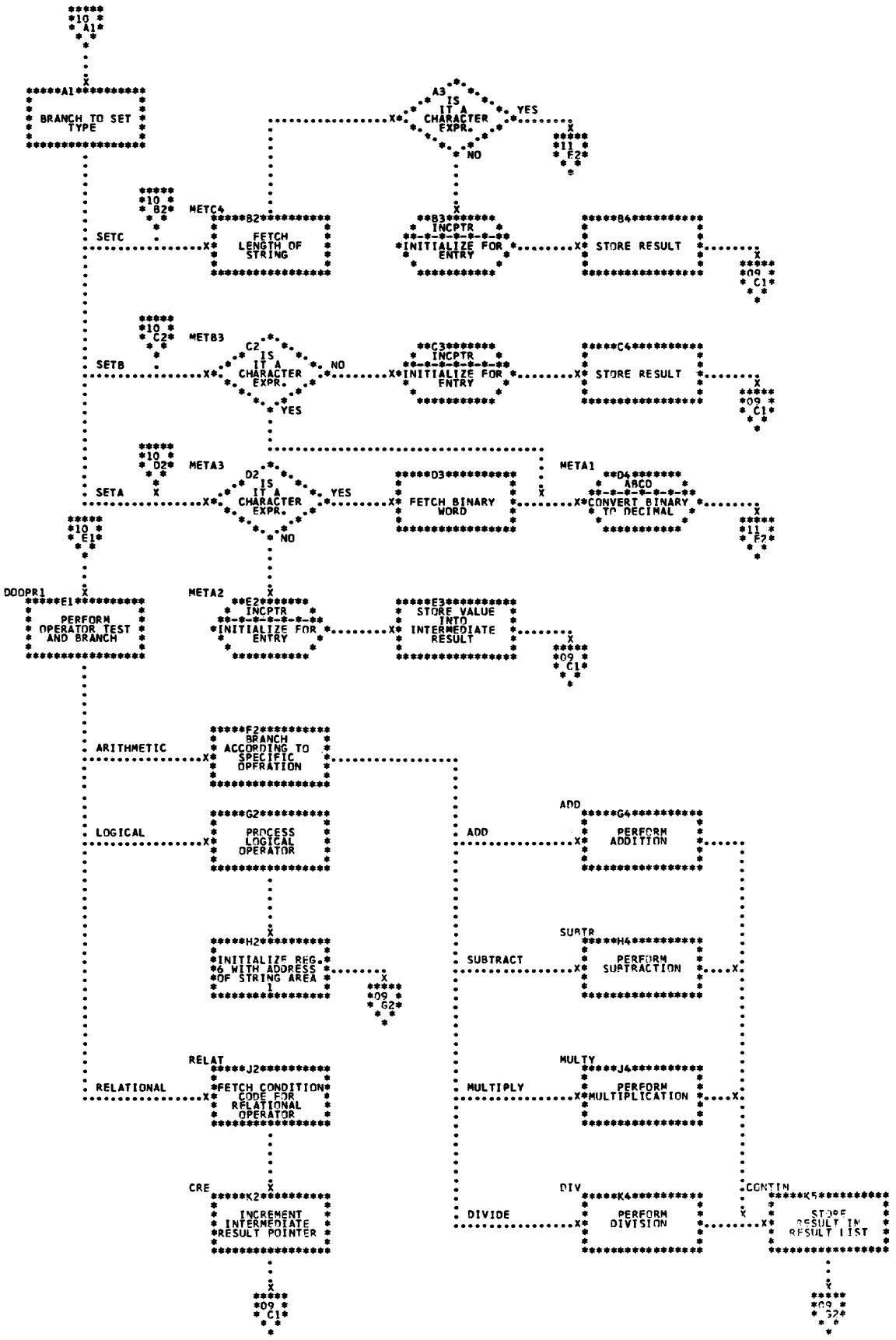



Chart 10. VALUAT - Phase F3 Evaluation (2 of 3)

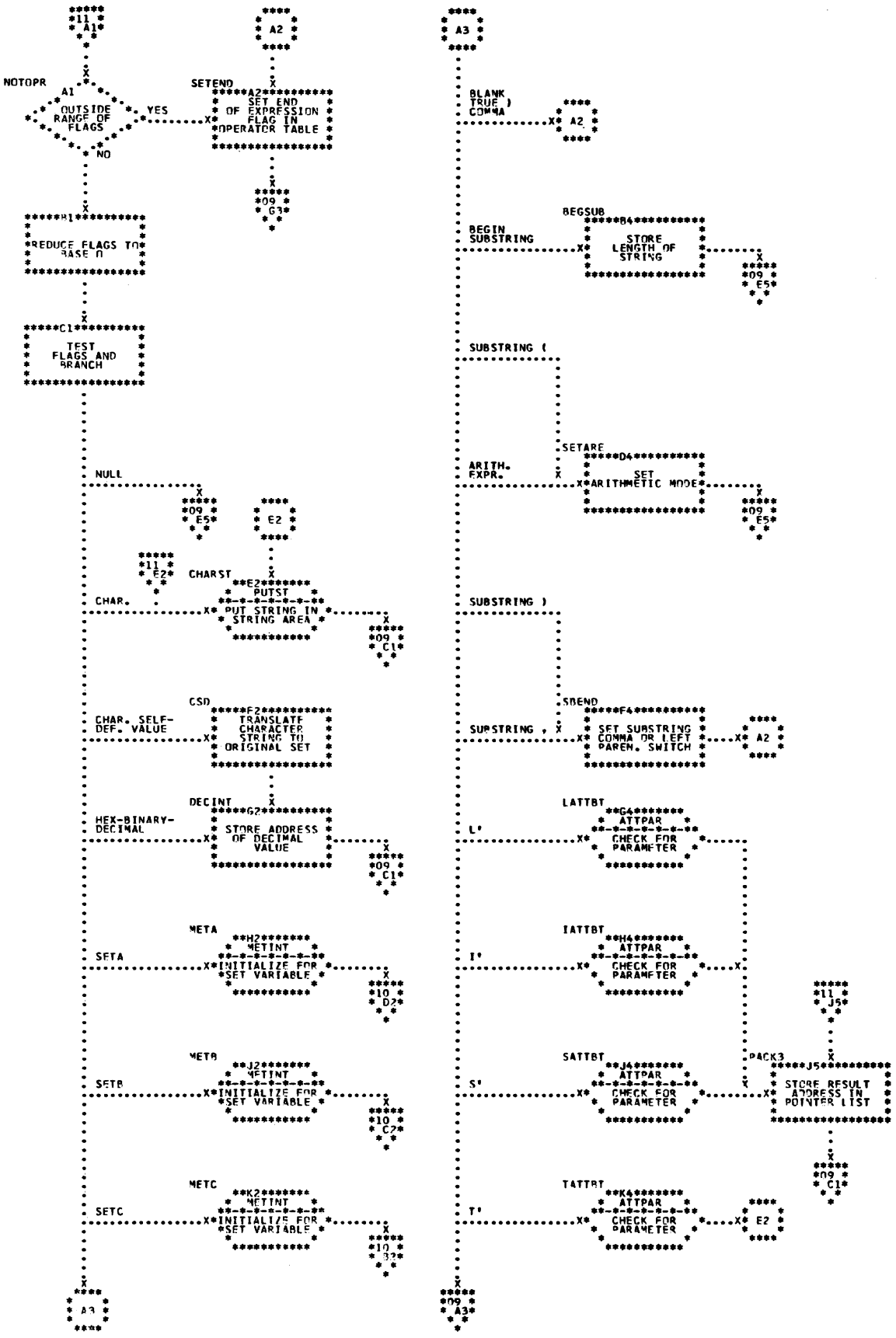



Chart 11. VALUAT - Phase F3 Evaluation (3 of 3)

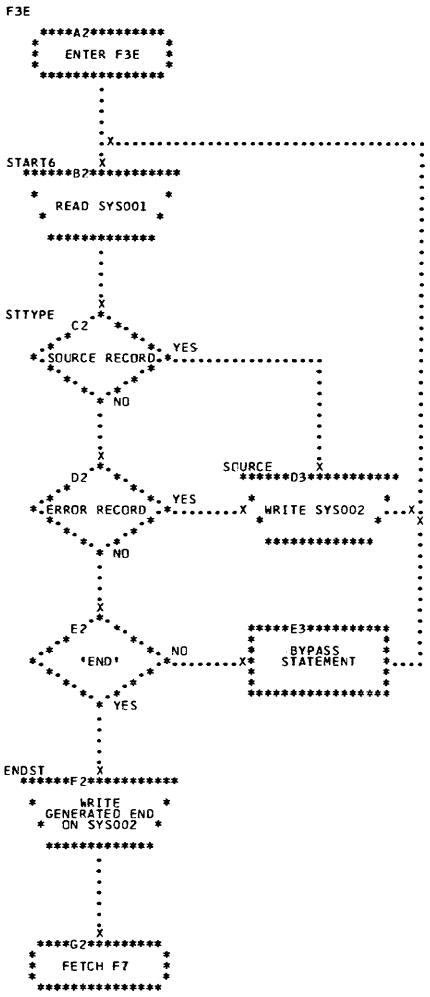



Chart 12. F3E - Phase F3 Substitute

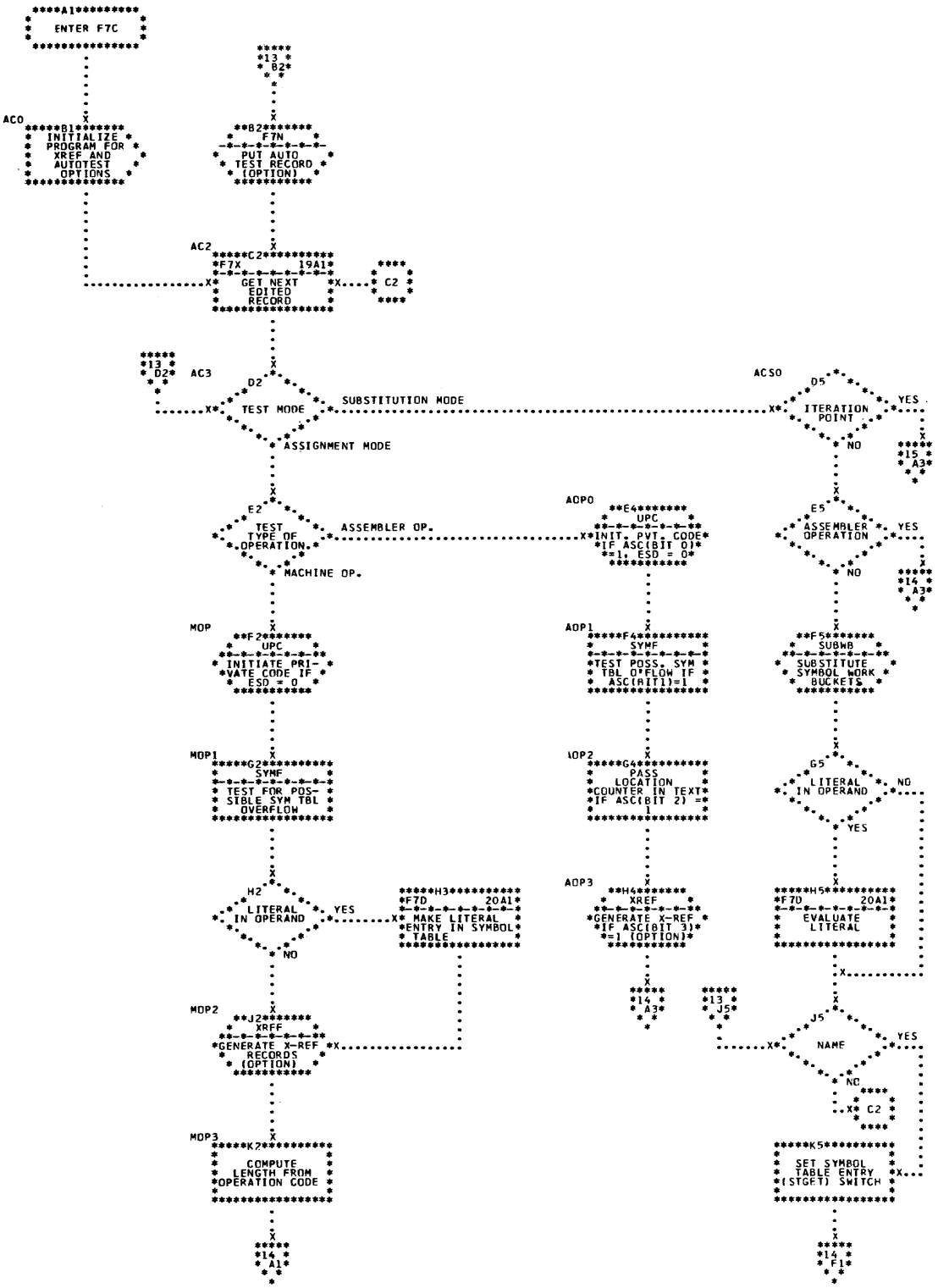



Chart 13. F7C - Phase F7 Mainline Control (1 of 6)

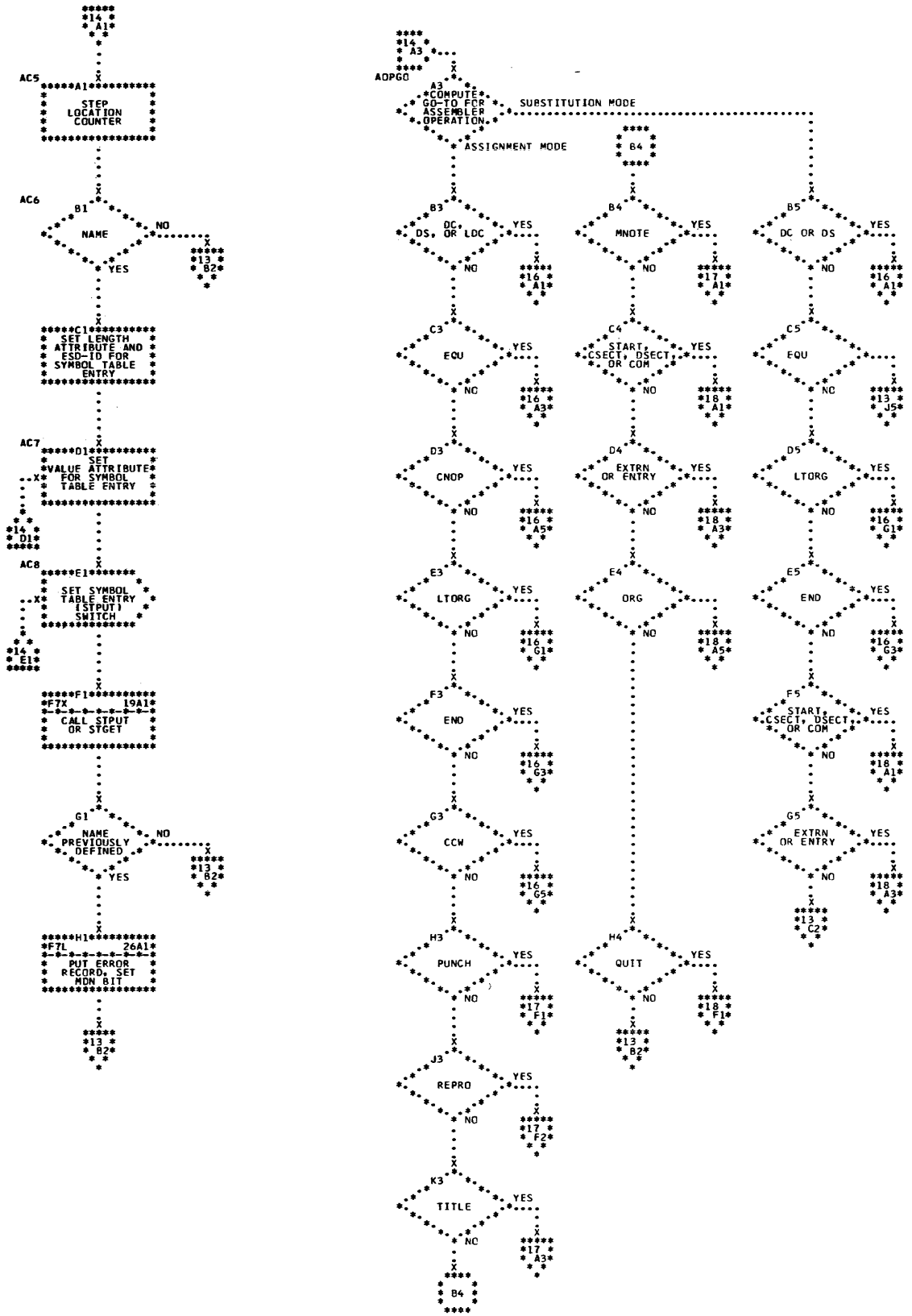



Chart 14. F7C - Phase F7 Mainline Control (2 of 6)

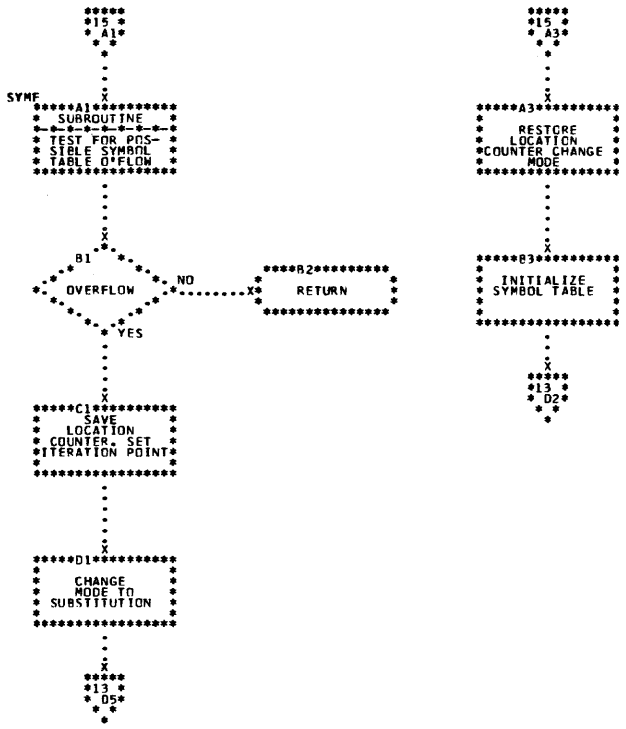



Chart 15. F7C - Phase F7 Mainline Control (3 of 6)

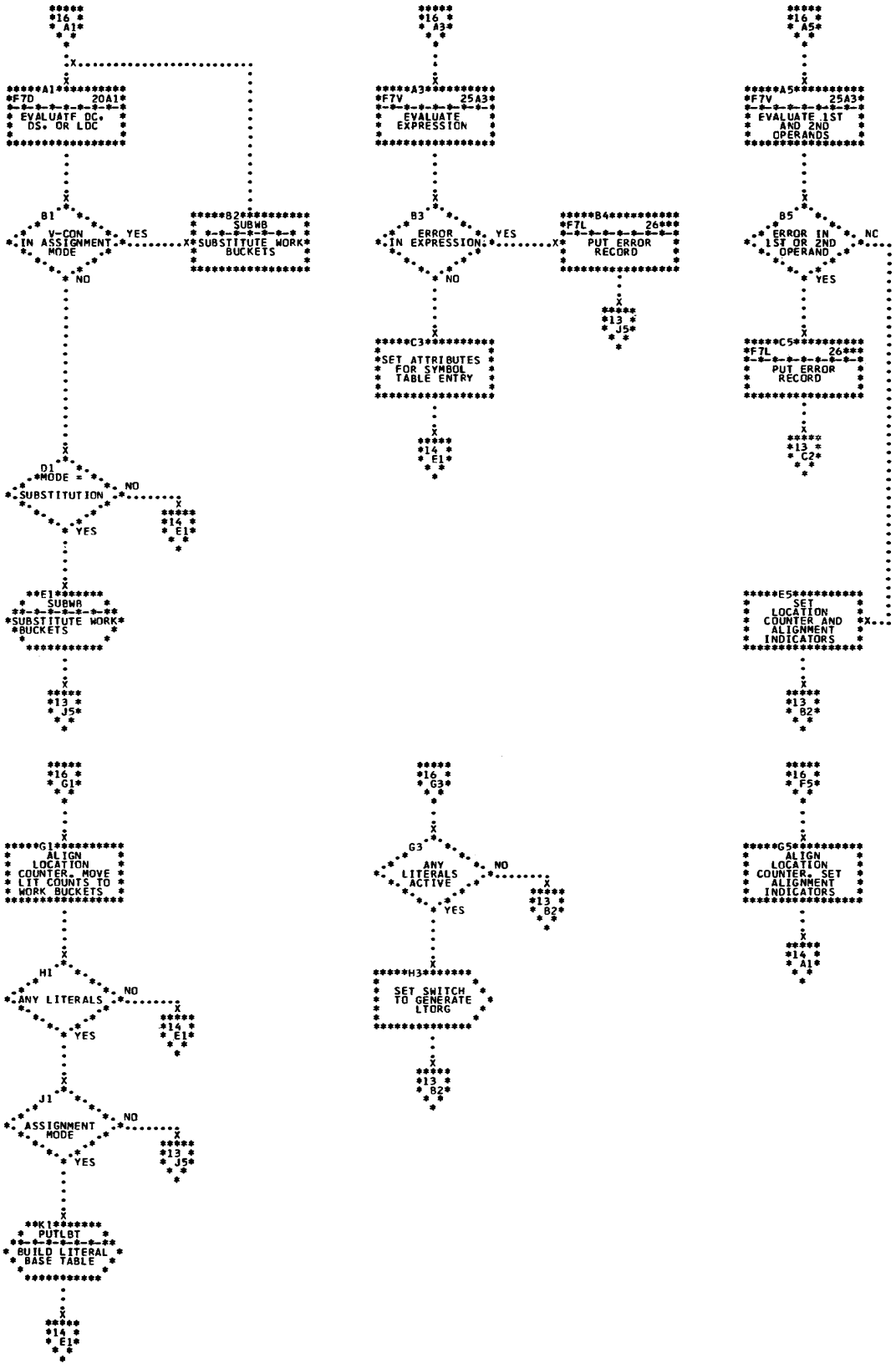



Chart 16. F7C - Phase F7 Mainline Control (4 of 6)

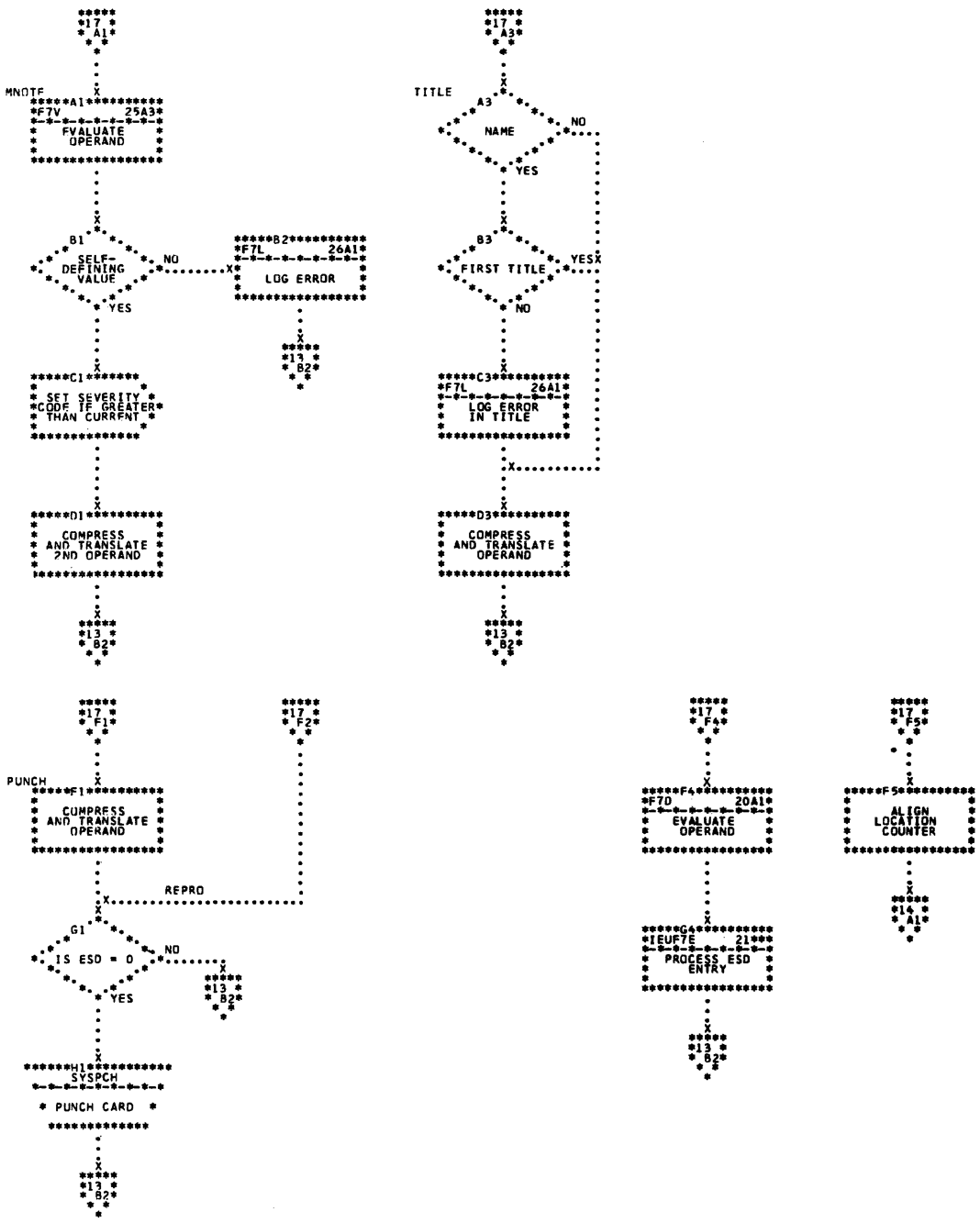



Chart 17. F7C - Phase F7 Mainline Control (5 of 6)

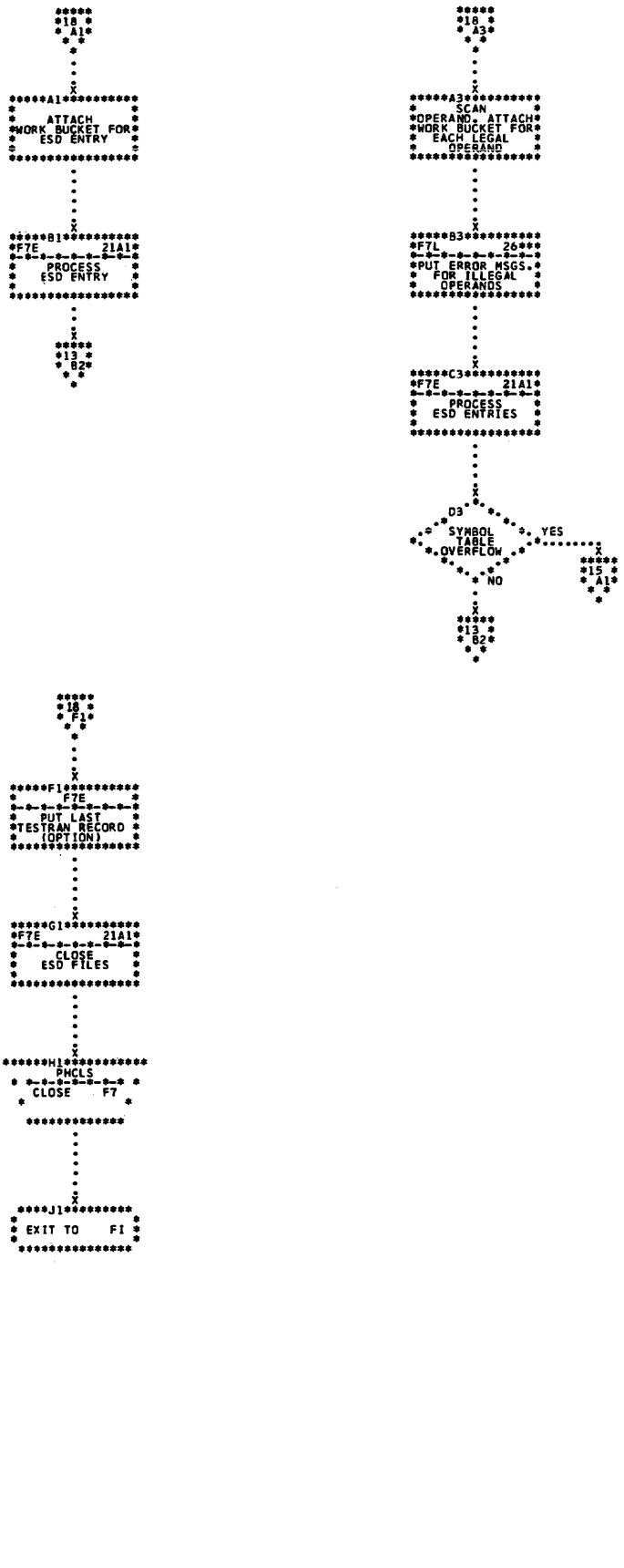



Chart 18. F7C - Phase F7 Mainline Control (6 of 6)

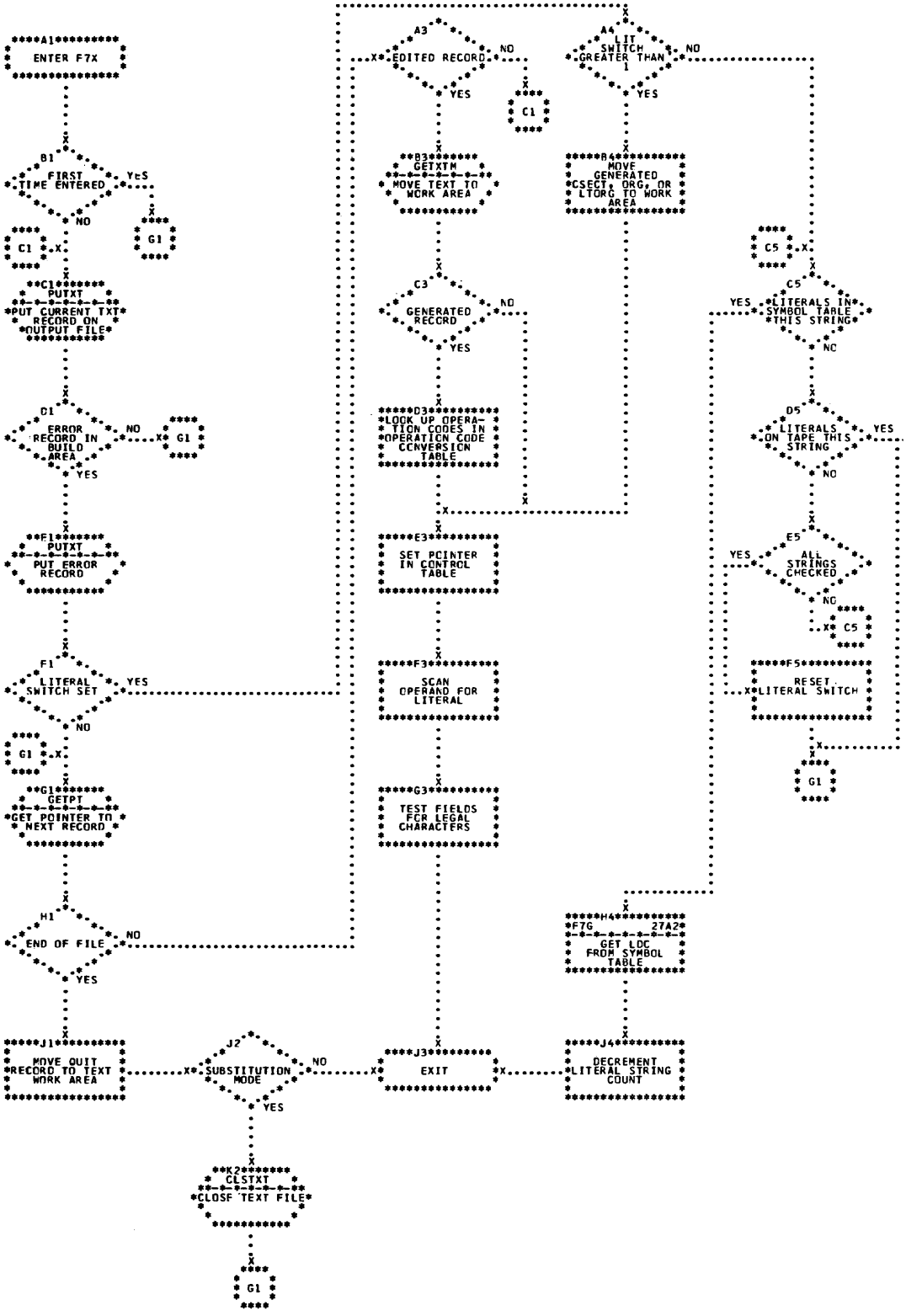



Chart 19. F7X - Phase F7 Get Statement

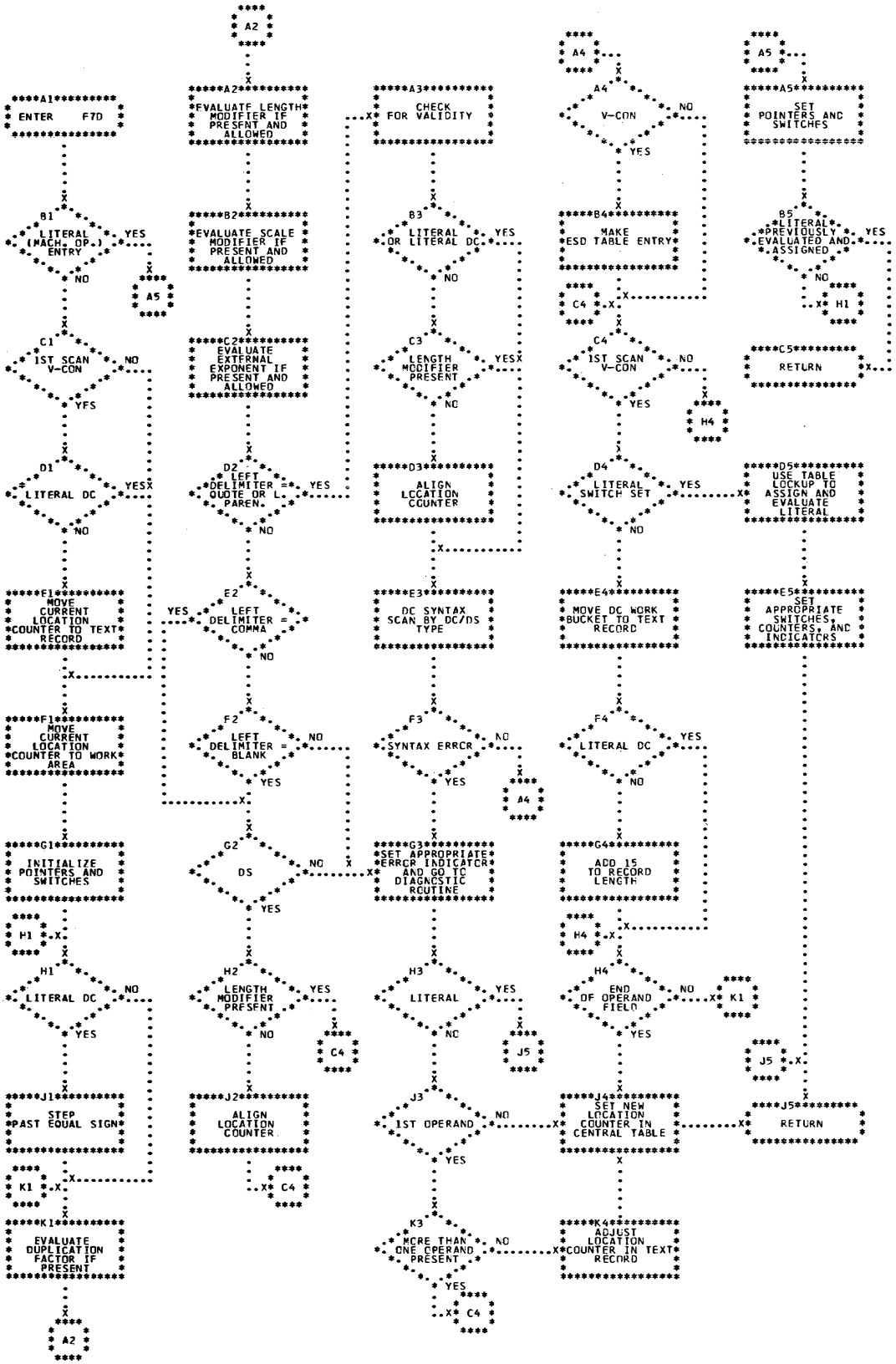



Chart 20. F7D - Phase F7 DC Evaluation

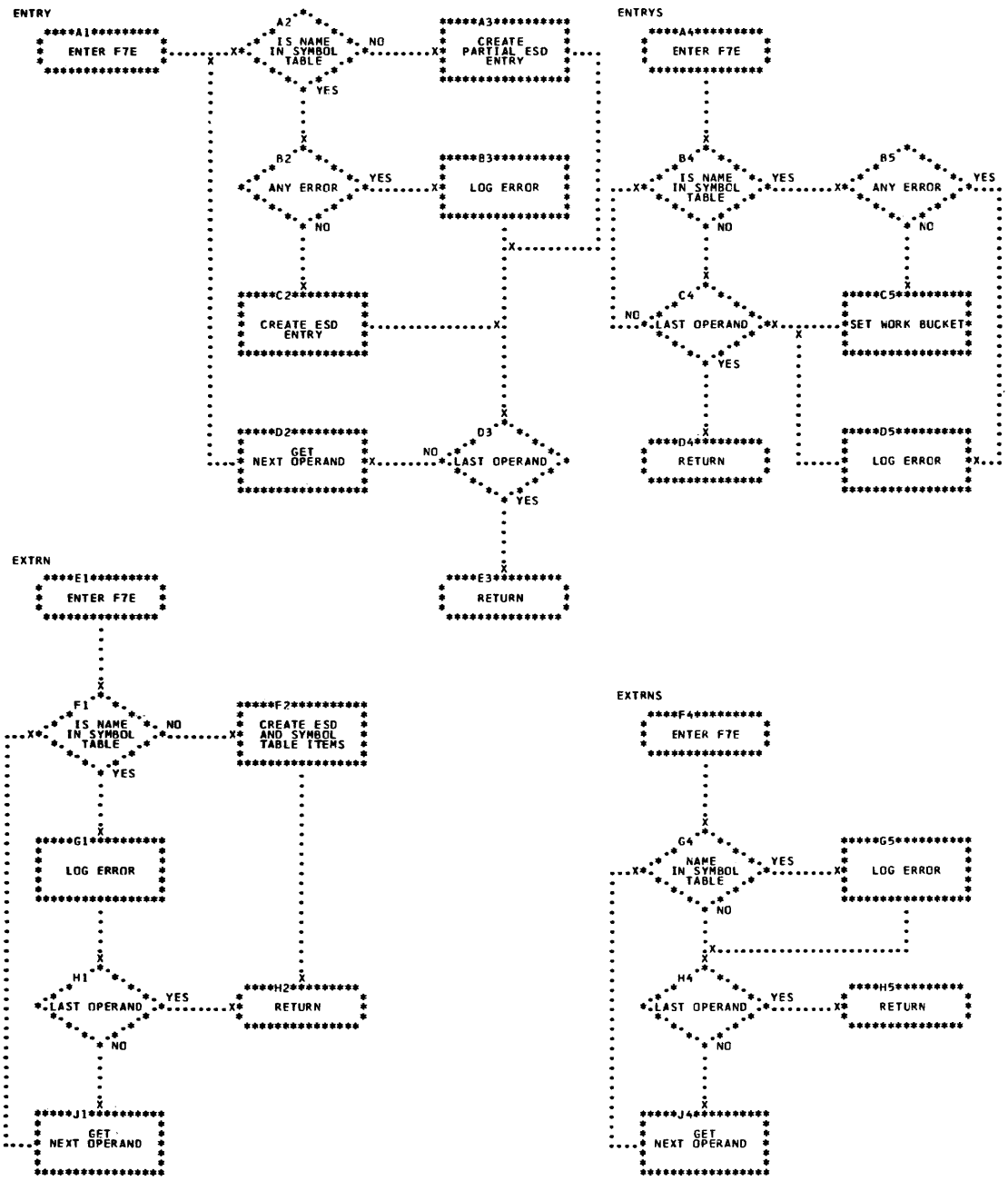



Chart 21. F7E - Phase F7 ESD Routine (1 of 3)

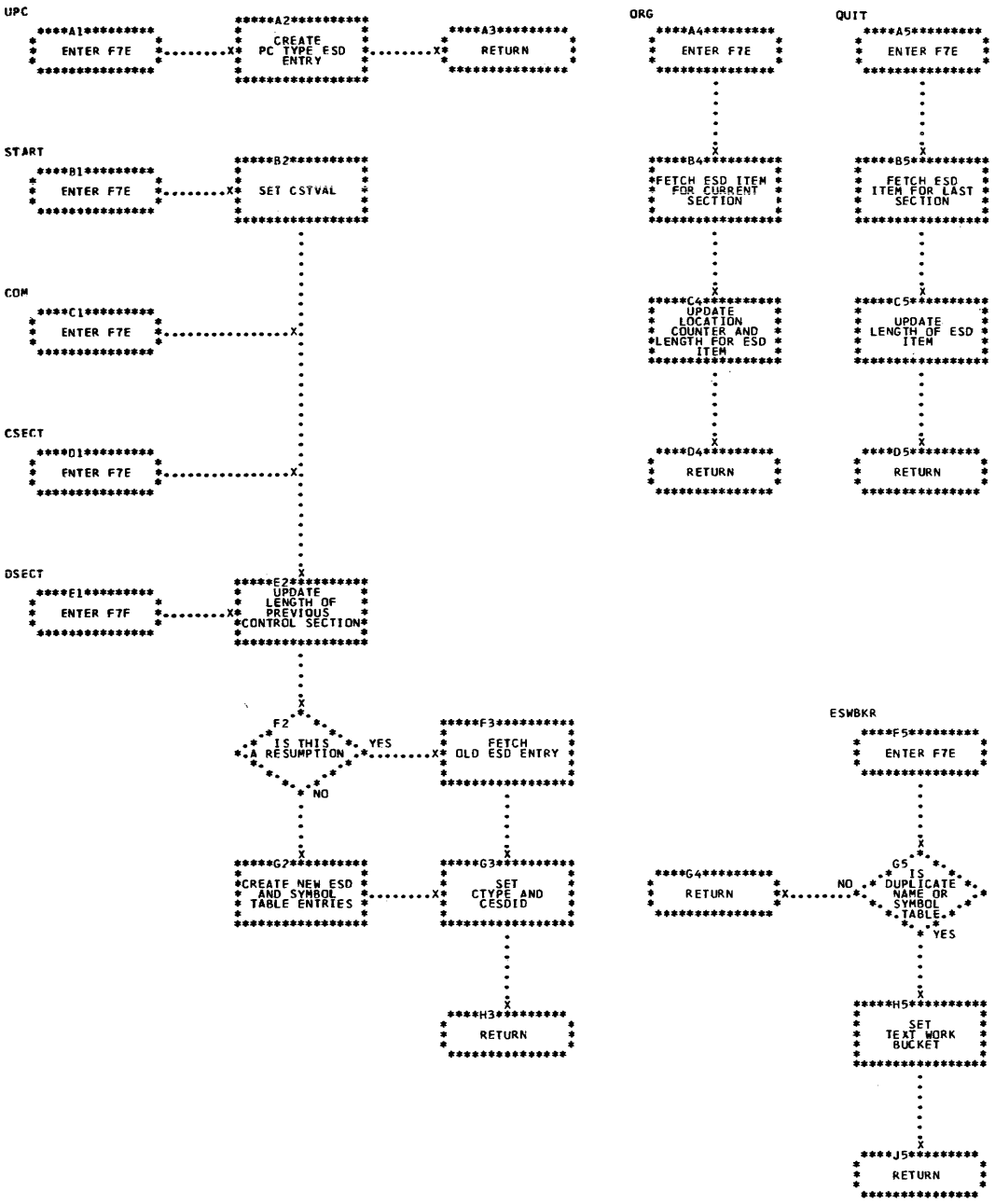



Chart 22. F7E - Phase F7 ESD (2 of 3)

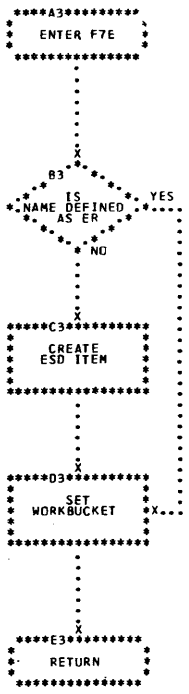



Chart 23. F7E - Phase F7 ESD Routine (3 of 3)

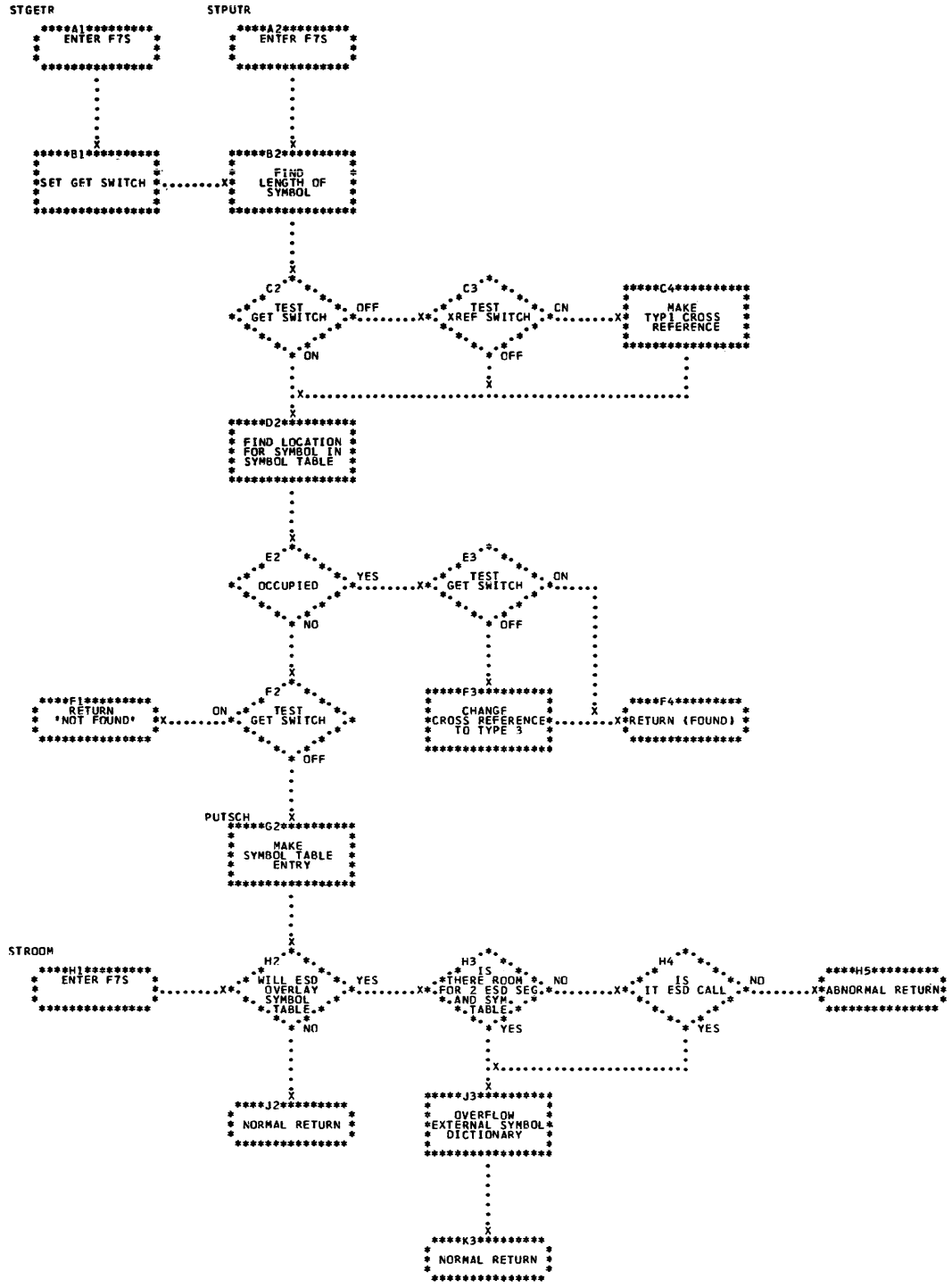



Chart 24. F7S - Phase F7 Symbol Table Routine

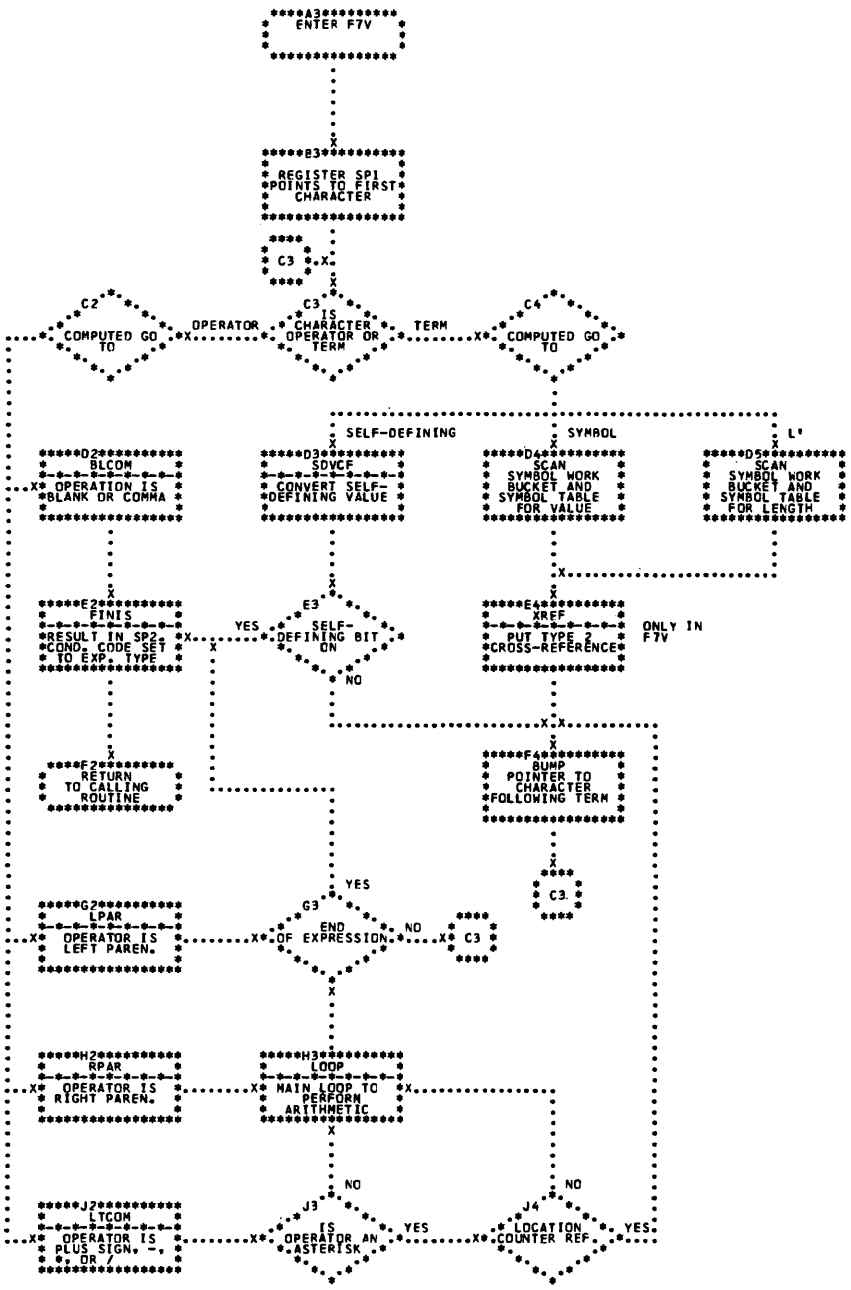



Chart 25. F7V - Phase F7 Expression Evaluation

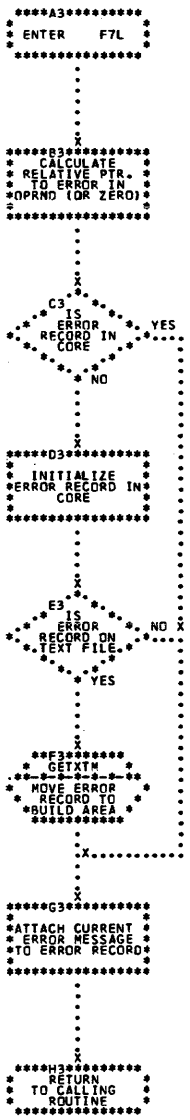



Chart 26. F7L - Phase F7 Log Error Routine

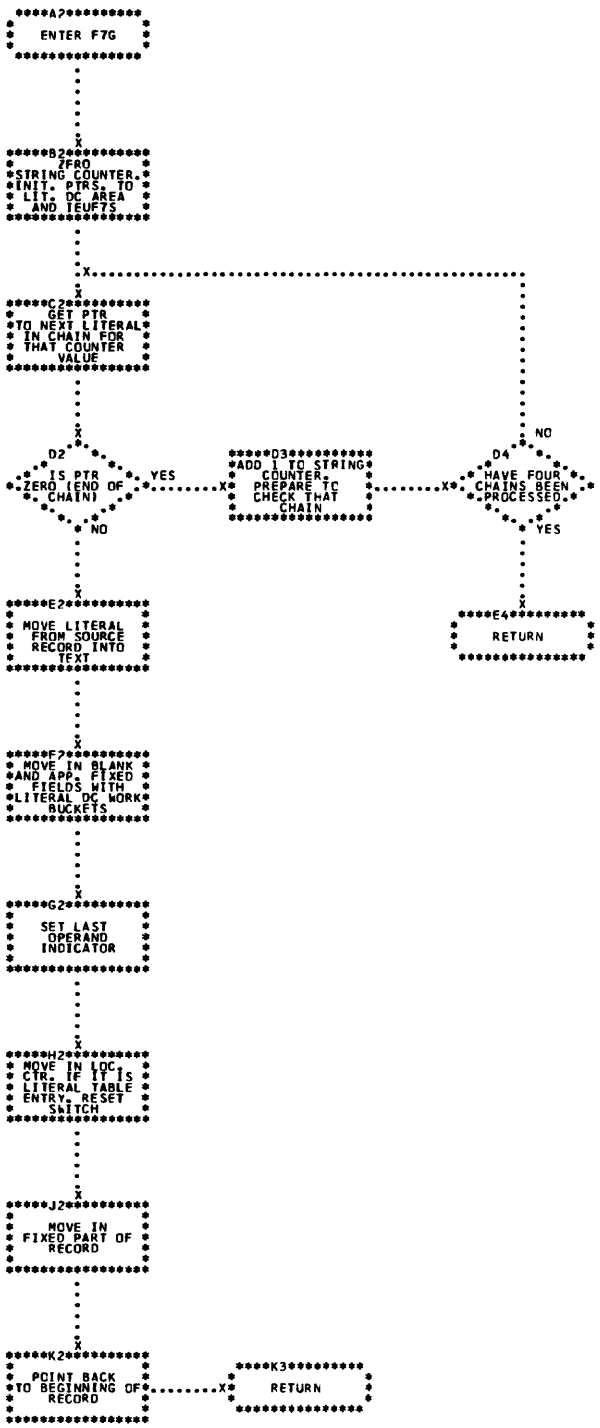



Chart 27. F7G - Phase F7 DC Get Routine

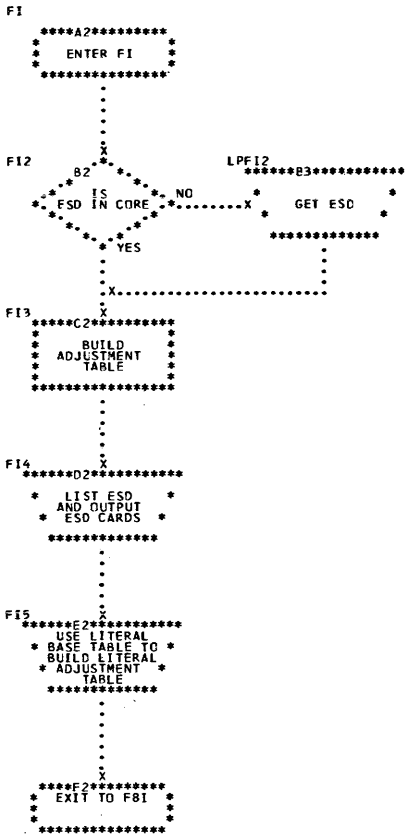



Chart 28. FI - Phase F Interlude

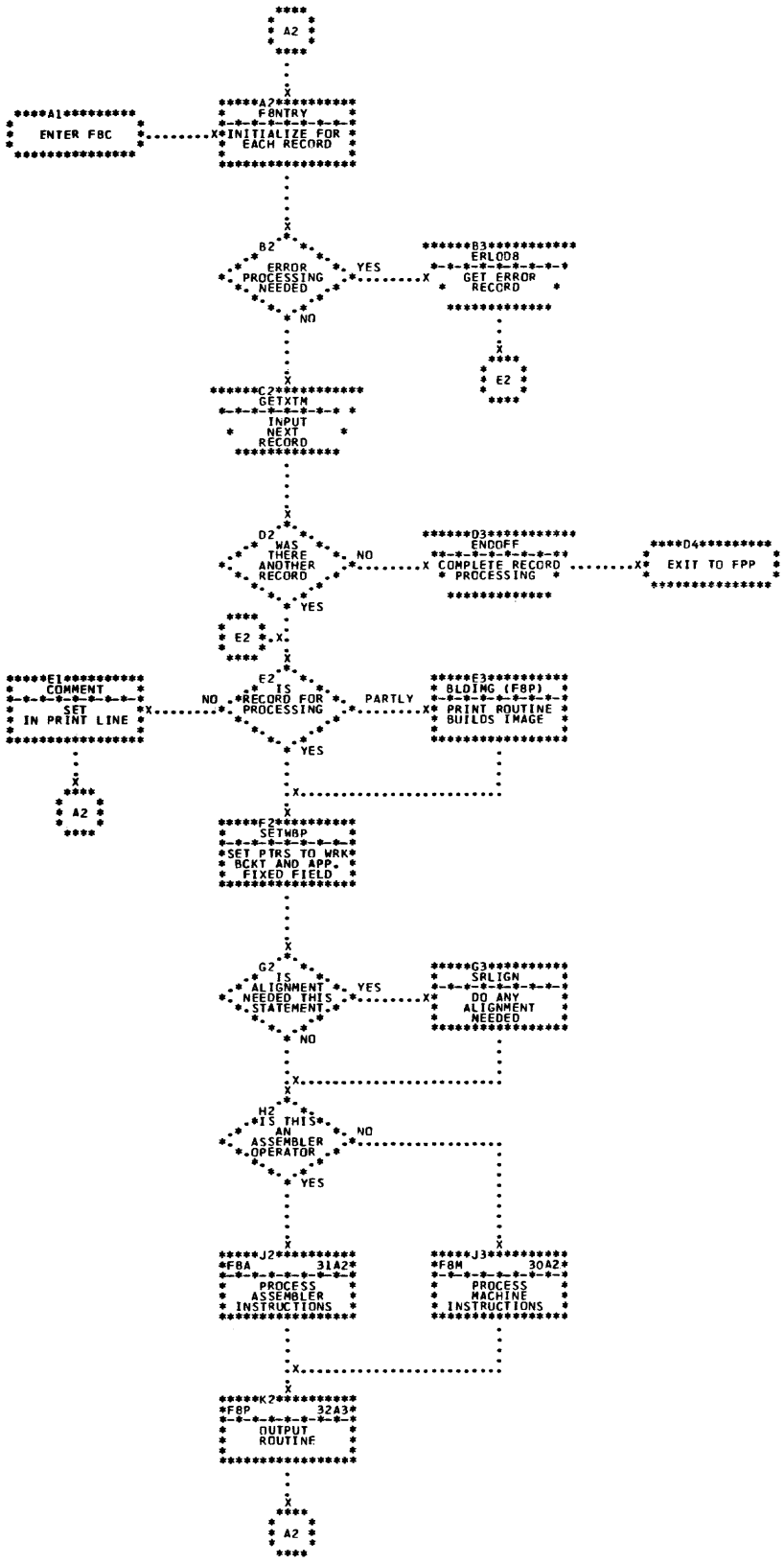



Chart 29. F8C - Phase F8 Mainline Control

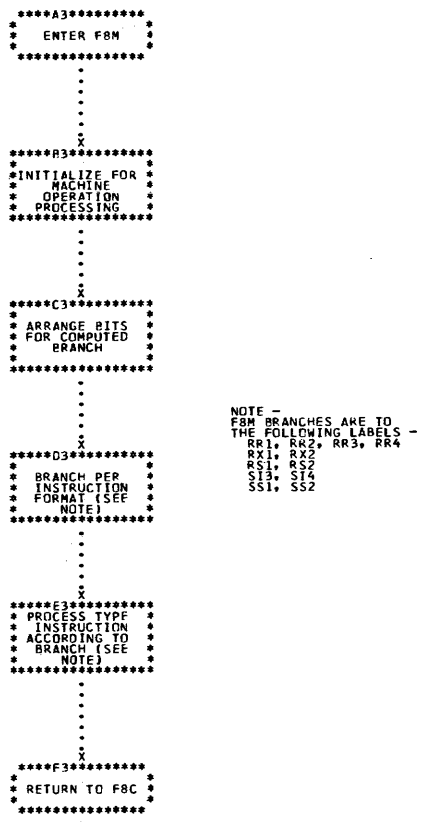
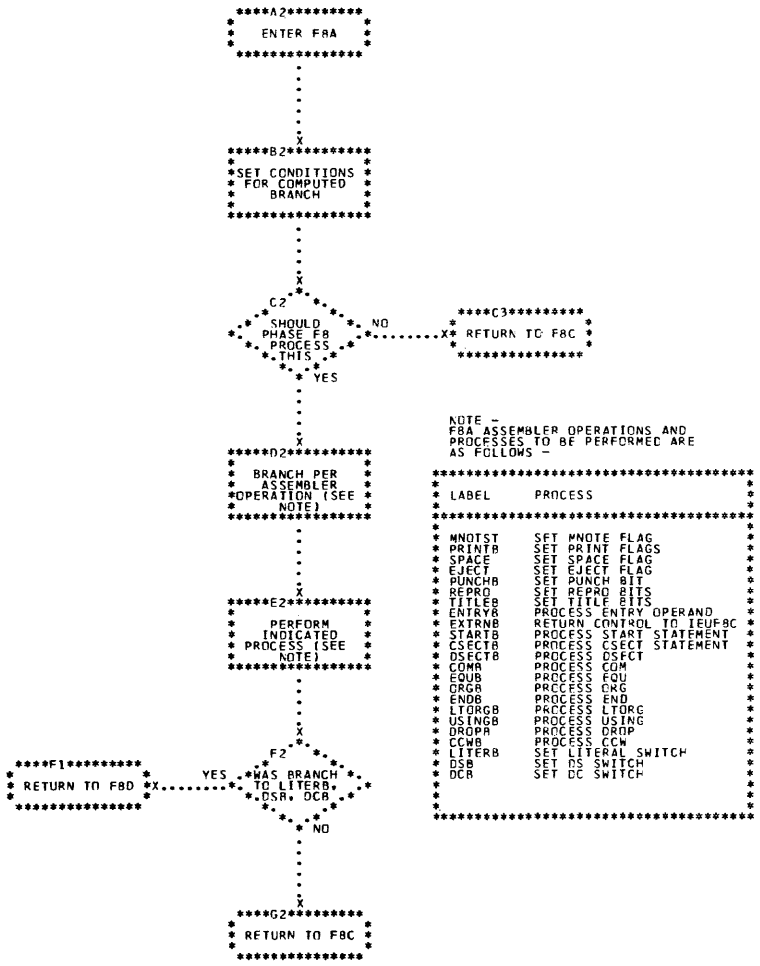



Chart 30. F8M - Phase F8 Machine Operation Processor




Flowchart 31. F8A - Phase F8 Assembler Operation Processor

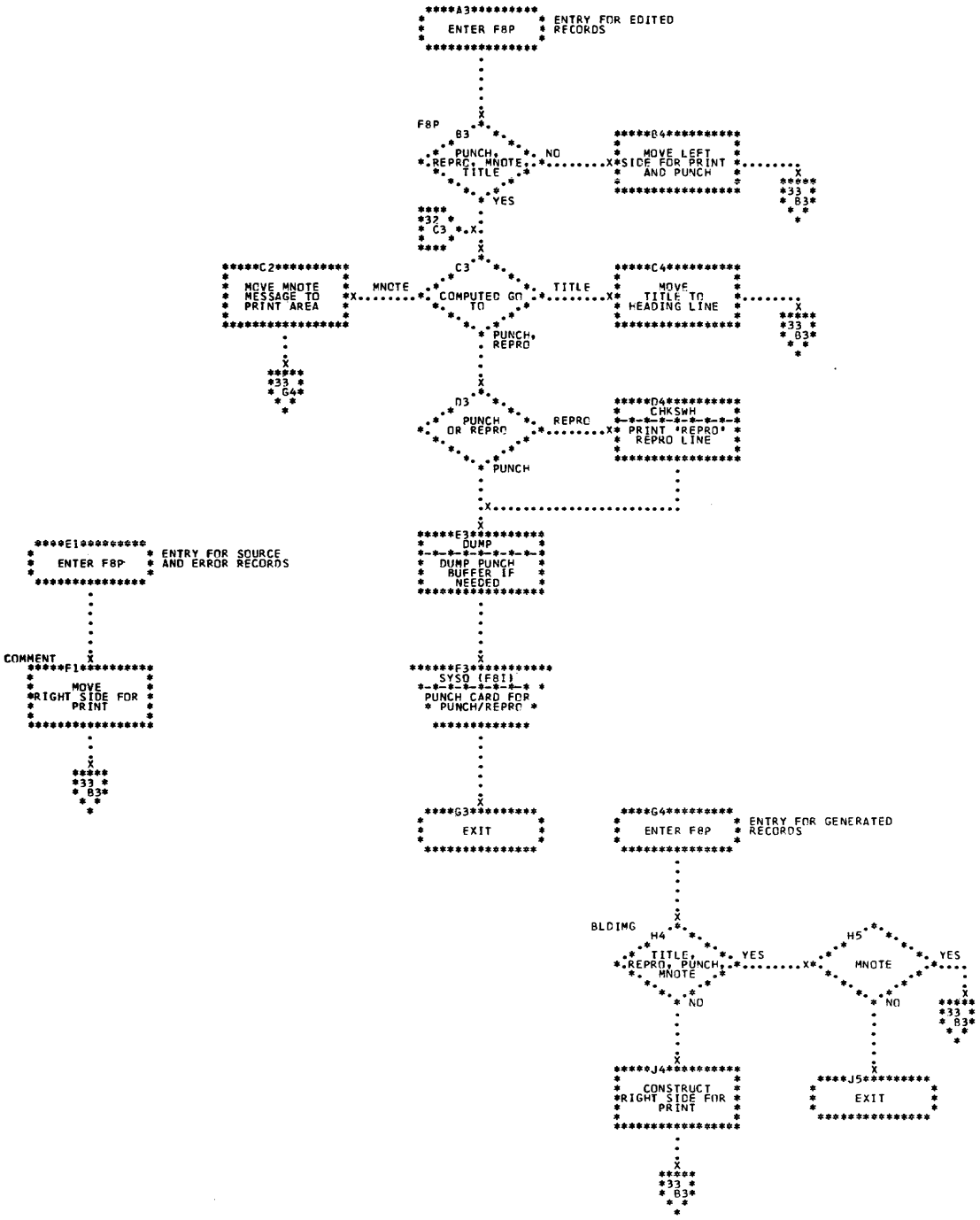



Chart 32. F8P - Phase F8 Print Routine (1 of 2)

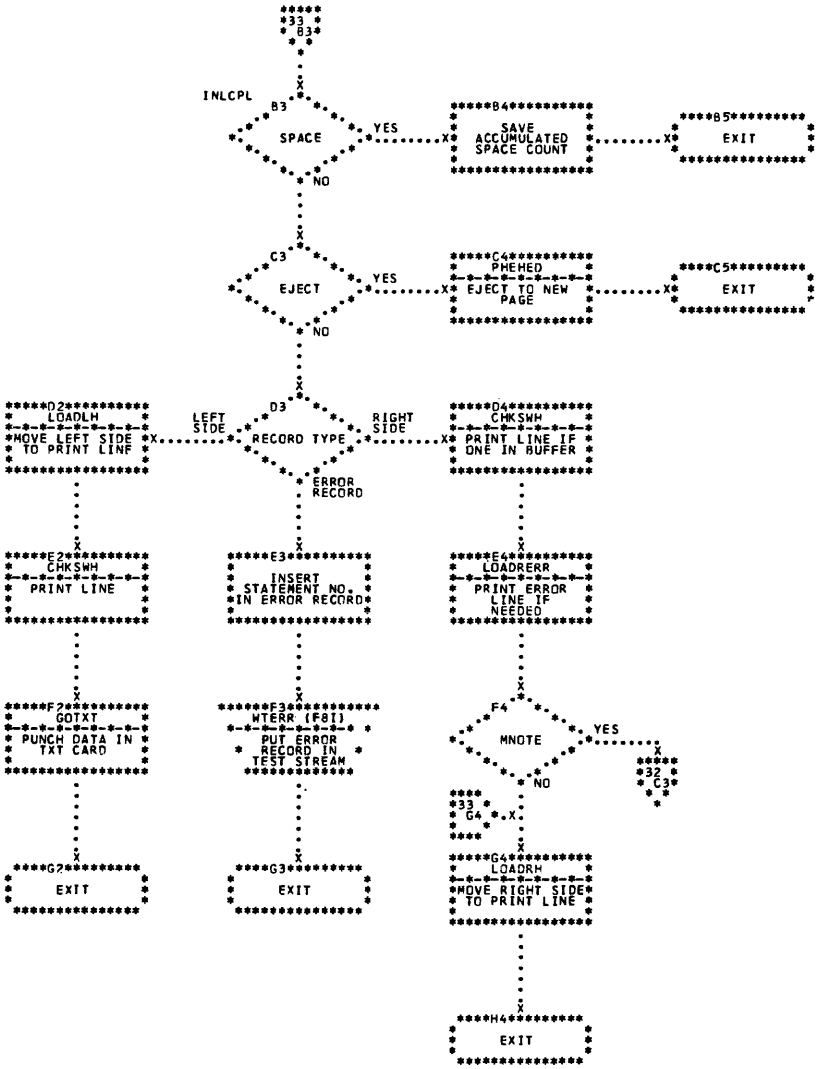



Chart 33. F8P - Phase 8 Print Routine (2 of 2)

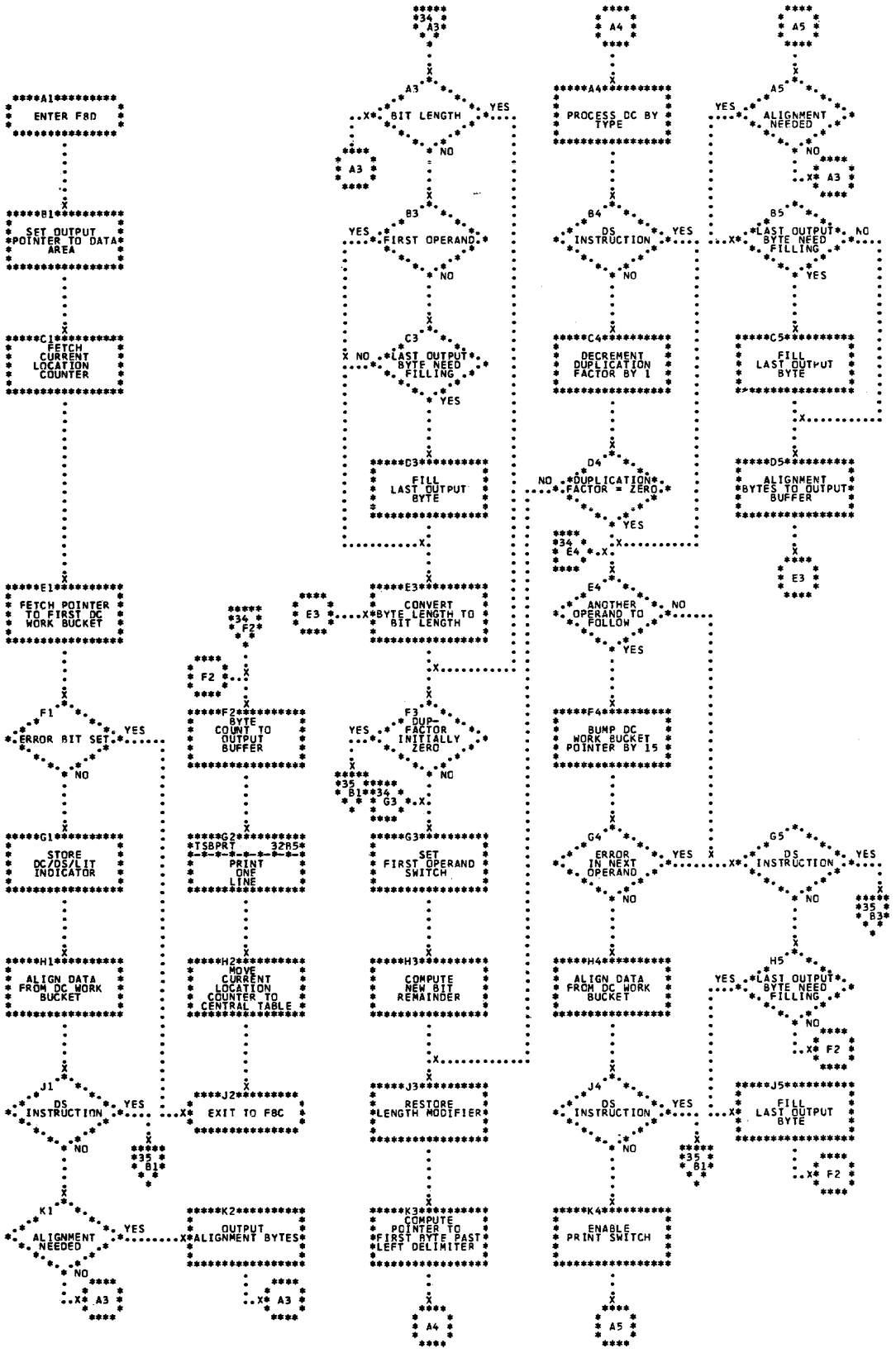



Chart 34. F8D - Phase F8 DC Evaluation (1 of 2)

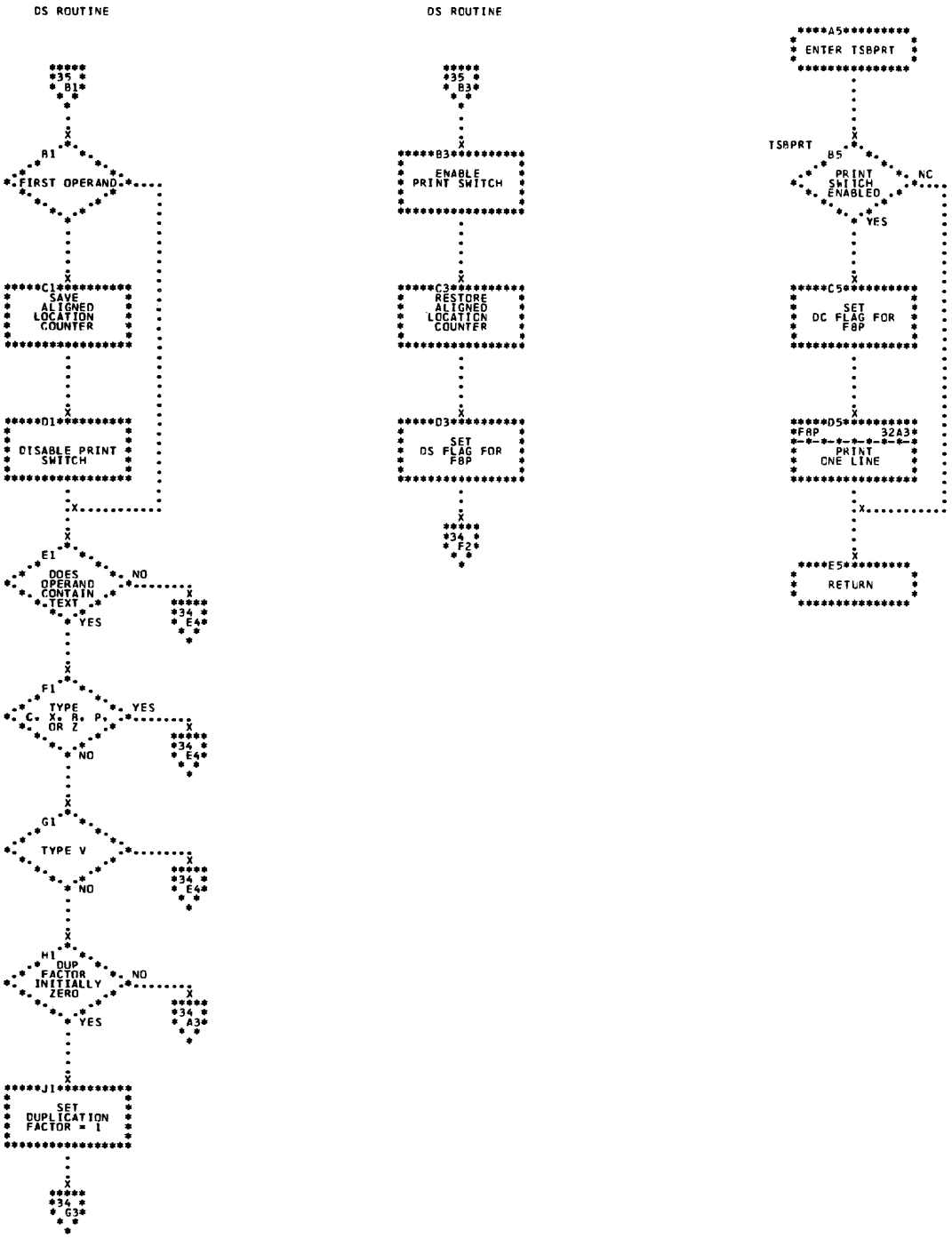



Chart 35. F8D - Phase F8 DC Evaluation (2 of 2)

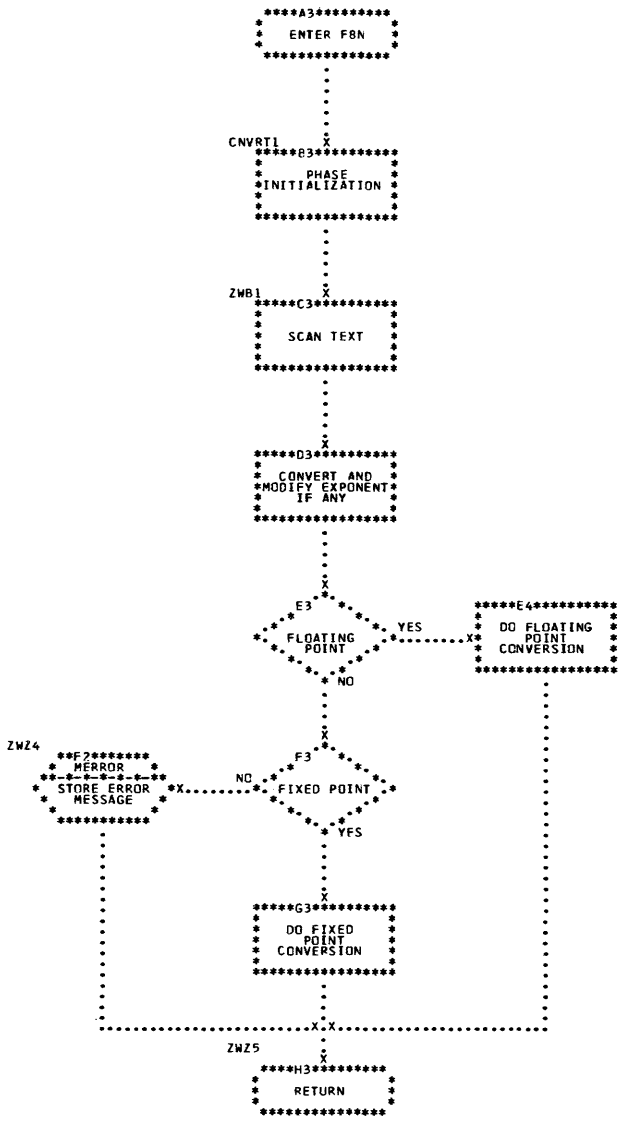



Chart 36. F8N - Foating and Fixed Point Conversion

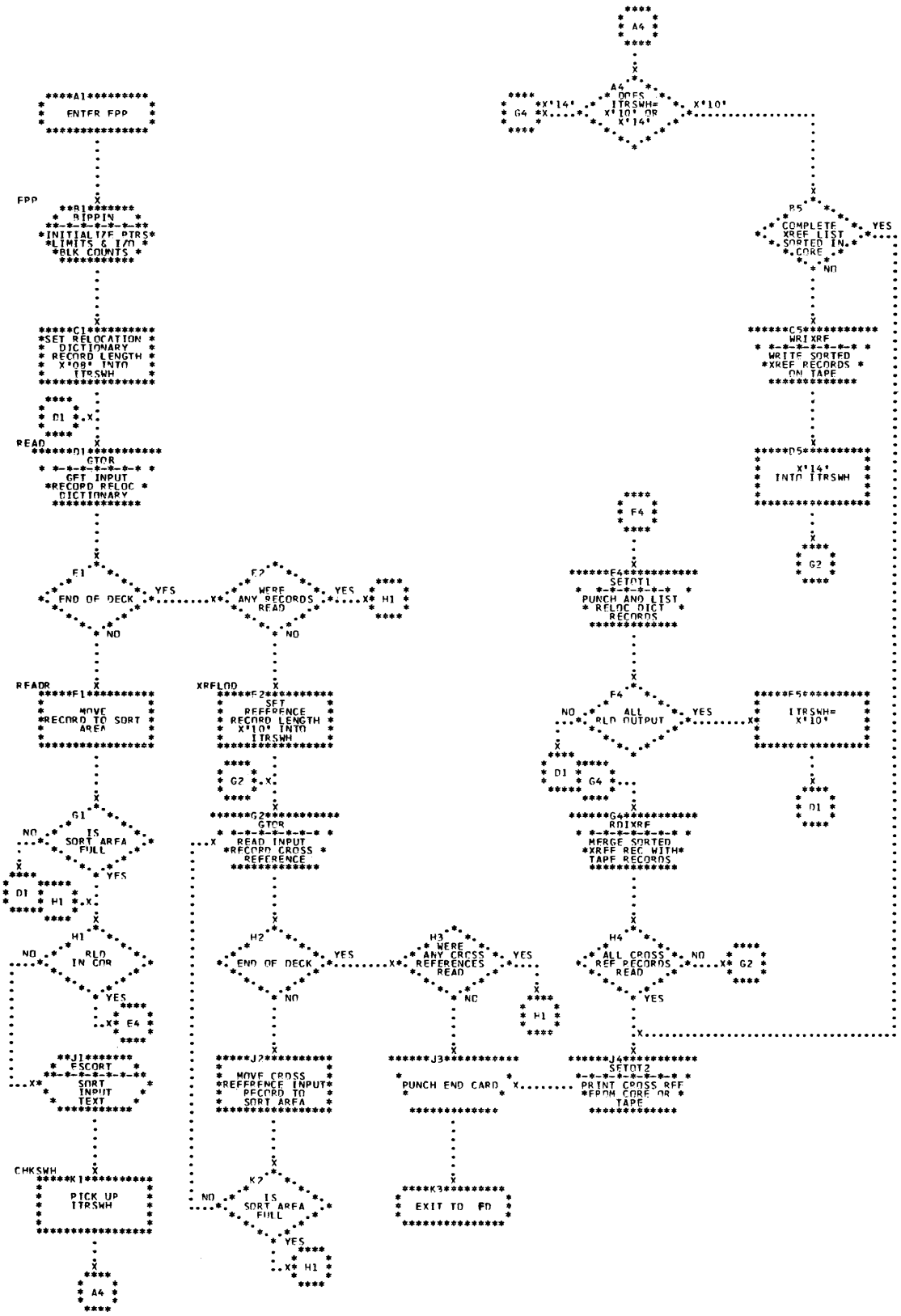



Chart 37. FPP - Phase FPP Post Processor

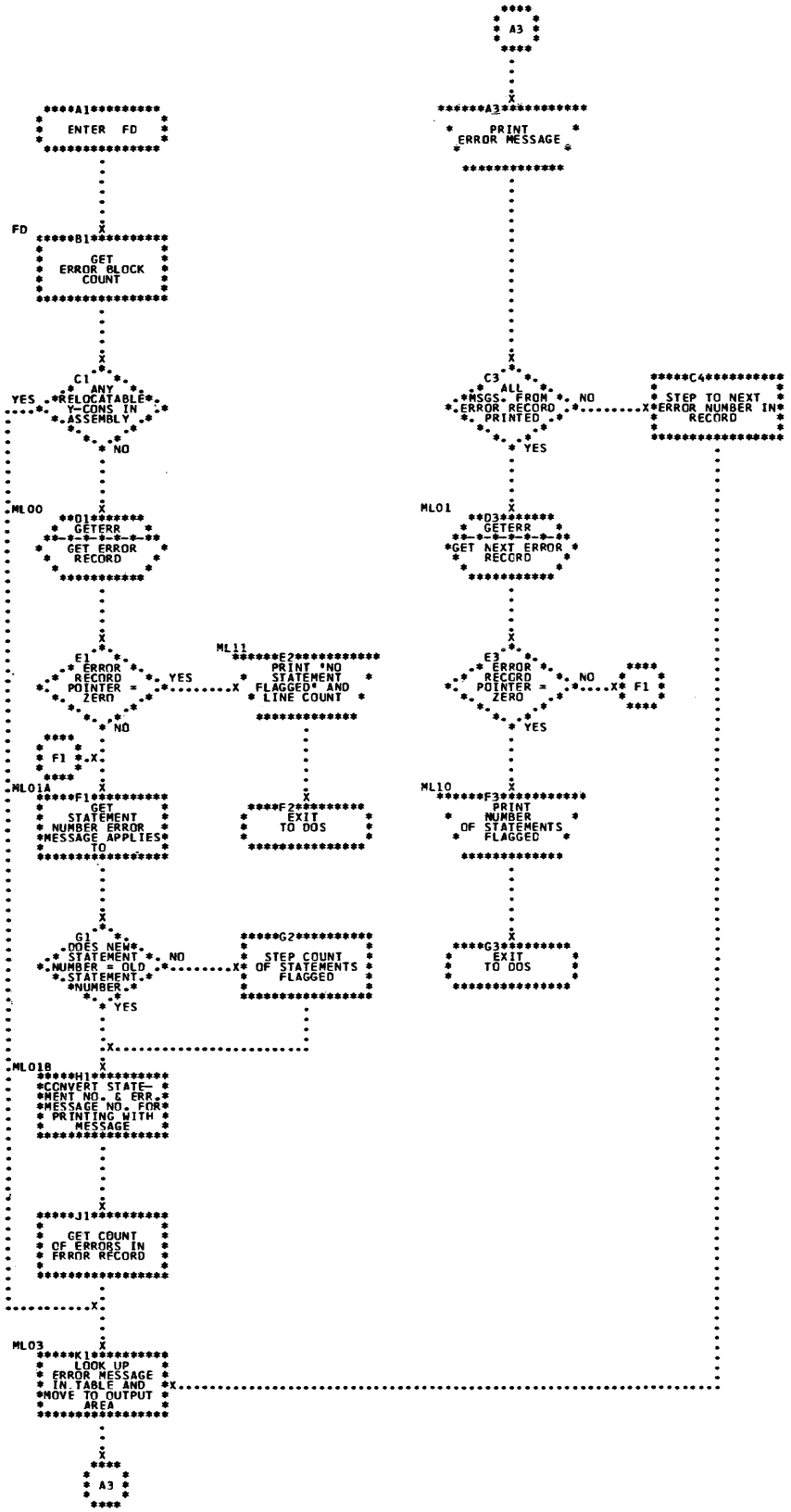



Chart 38. FD - Phase FPP Diagnostic

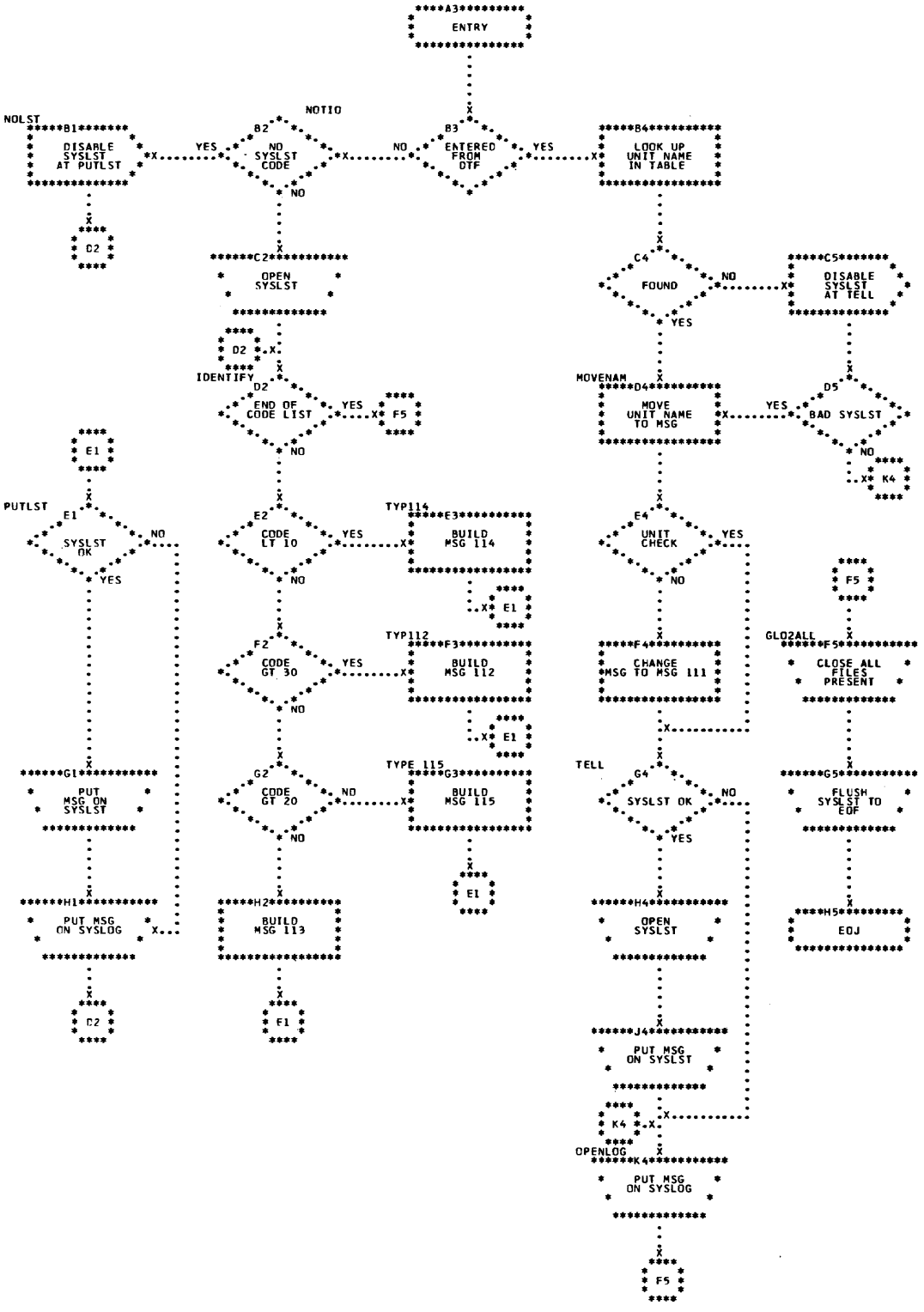



Chart 39. ABT - Phase ABORT

OPTIONS

The programmer may specify the following assembler options in the // OPTION card.

```
      DECK                LIST   SYM   XREF
NODECK,LINK,CATAL,NOLIST,NOSYM,NOXREF
```

These options are defined as follows:

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

LINK or CATAL - The object module is placed on the device specified in the SYSLNK statement. In addition, CATAL causes the core image module produced by LNKEDT to be cataloged on the core image library.

LIST - An assembler listing is produced.

SYM - The object module (if produced) contains the special symbol table required by the test translator (AUTOTEST) routines.

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

The prefix NO is used with the above options to indicate that the option is not wanted. If contradictory options are entered, e.g., LIST, NOLIST, the rightmost option, e.g., NOLIST, is used.

DEFAULT ENTRY

If no options are specified, the assembler assumes the default entry values supplied at SYSGEN.

## APPENDIX B. DICTIONARY, TABLE, AND RECORD FORMATS

The dictionary, table, and record formats are grouped according to phase. The macro generation phases create dictionaries (global and local) and source, error, and edited text records.

The assembly phases create other tables and dictionaries (symbol table, relocation dictionary, etc.,) and error records. They also create source records for generated statements and pass and/or modify the edited text records which are not resolved during macro generation and conditional assembly.

This appendix is divided by phases. The formats are then subdivided according to dictionary, table, and record formats within each phase.

### MACRO GENERATION PHASES (F2 AND F3)

Phase F2 creates and subsets the global, open code local, and macro local dictionaries. The subsetted dictionaries are used by Phase F3 for macro definition editing and conditional assembly.

F2 creates source records for text stream statements from SYSIPT. F2 also creates edited text records for all input statements from SYSIPT and SYSSLB -- except it creates no edited text records for ICTL, ISEQ, MACRO, COPY, or comments (* and .*) statements. Edited text records are either Type I or Type II. Type I records include machine instructions and all assembler instructions except Type II. Type II records have non-standard formats. They include REPRO, SETx, AIF, AGO, MEND, MEXIT, and End-of-Data-Set records. Type II records (except REPRO) are resolved during F3. Type I records and REPRO records are modified by F3 for processing by the assembly phases.

#### Macro Generator Dictionary Entries

The macro generator dictionaries -- illustrated in Figure B1 -- are summarized below.

Global and Local Dictionaries

"A" Pointer (global big "A" pointer) - Forward chaining address of the next entry in a dictionary chain.

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

Flag - (Global Dictionaries)

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 code
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)

Flag - (Local Dictionaries)

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)

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

Mnemonic - The name in Internal Assembler Code.

"a" Pointer - Little "a" pointer to a location in the Phase F3 dictionary that

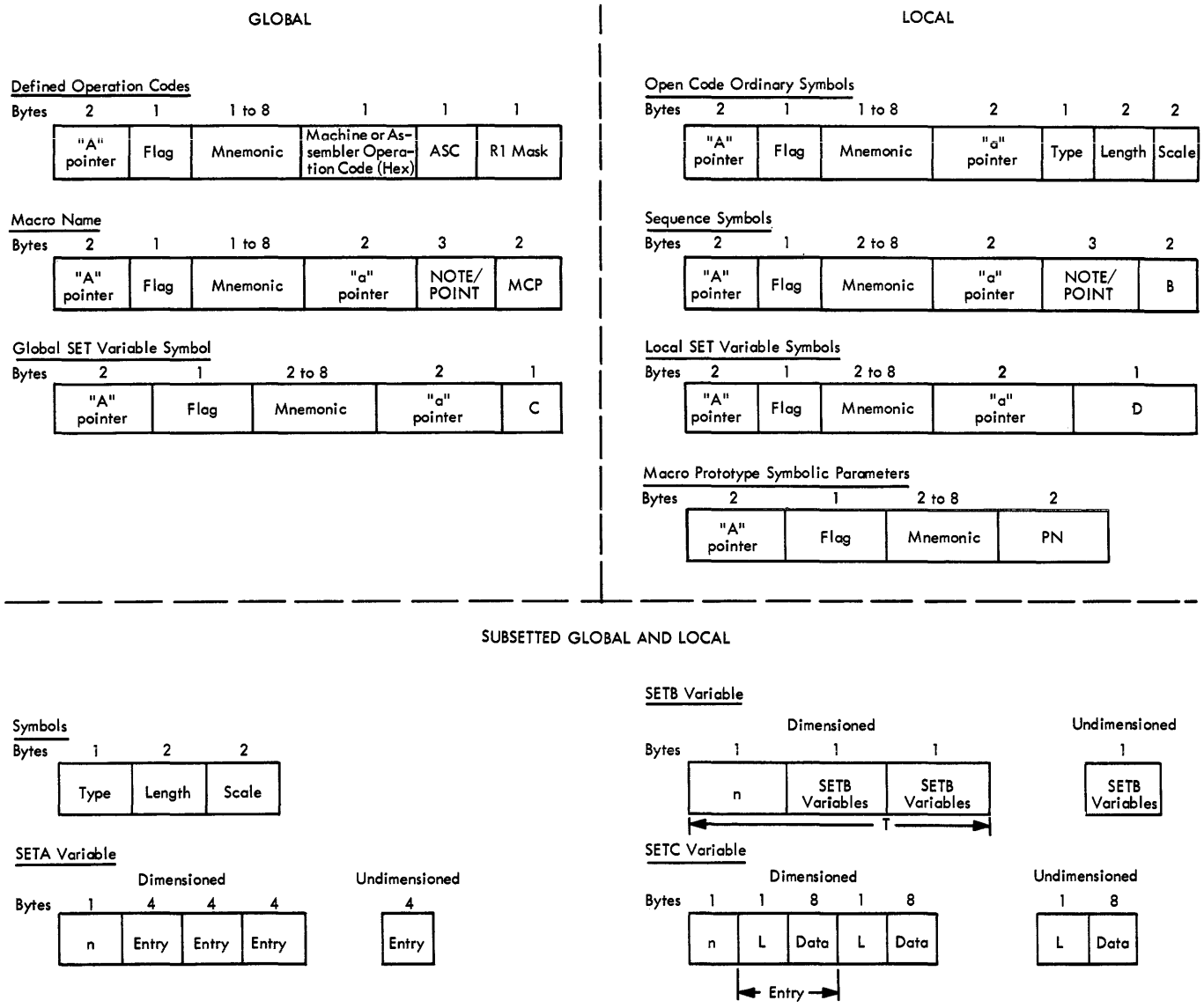



Figure B1. Macro Generator Dictionary Entry Formats

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 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 edited text of the macro definition.

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

B - The position of the beginning of the sequence symbol edited text.

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

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

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

Type - Type attribute. See Table B1.

Length - Length attribute.

Scale - Scale attribute:

Bit 0 0 - Positive
1 - Negative

Bits 1-15 Scale attribute

Subsetted Dictionaries

n - number of variables in entry

L - True length of character string (data)

T - Length of SETB dictionary entry (n/8+1)

Type, Length, Scale - See above.

Macro Local Dictionary Header

| Bytes | 4 | 4 | 4 | 4 | 1 | 1 | 2 |
|---|---|---|---|---|---|---|---|
| | Dummy | ACTR Loop | Dummy | NOTE/ POINT | Dummy | No. of blocks | Size of md. |

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

ACTR Loop - The ACTR loop limit, i.e., initial assumption made by processor.

NOTE/POINT - Location of block on SYS003 in which the macro definition edited text begins (points to prototype record).

No. of blocks - The number of segments on SYS003 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:

| Bytes | 4 | 4 | 4 | 4 | 1 | 1 | 2 |
|---|---|---|---|---|---|---|---|
| | Dictionary address | ACTR loop | ACTR counter | NOTE/ POINT | Flag | SYS001/3 | Delta M-I |

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.

SYS001/3 - A switch to indicate:

8 - Input is from SYS003
0 - Input is from SYS001

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

Macro Dictionary Parameter Table Entries

A macro dictionary parameter table is built from the macro instruction and macro prototype parameters when the macro definition is edited. The parameter table is written on SYS003 immediately following the macro local dictionary. The parameter formats are illustrated in Figure B2.

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

T = Type attribute

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

'c' = character flag — a character string follows

Tot. L = Total length of sublist entry

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

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

N' = Number of operands in a sublist

N' = 1 if the operand is a sublist

N' = 0 if the operand is omitted

Figure B2. Macro Dictionary Parameter Table Entries

## Macro Generator Record Formats

The fields which make up the macro generator records -- illustrated in Figure B3 -- are explained below.

Type ID - Type of record. See Table B1. This byte is dropped after Phase F3.

R/L - Record Length

FLAGA - A two byte field of information on how the record should be processed. See Edited Text Record Fixed Field Format in Phase F7. The source record contains only the first FLAGA byte.

Op Code - Absolute (hexadecimal) operation code for machine and assembler operations. See the DOS Assembler Language manual for the machine operation codes and Table B2 for assembler operation codes.

ASC - Assembler Switch Codes. (Inserted in F2 but not used by the macro generator

Table B1. 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 |

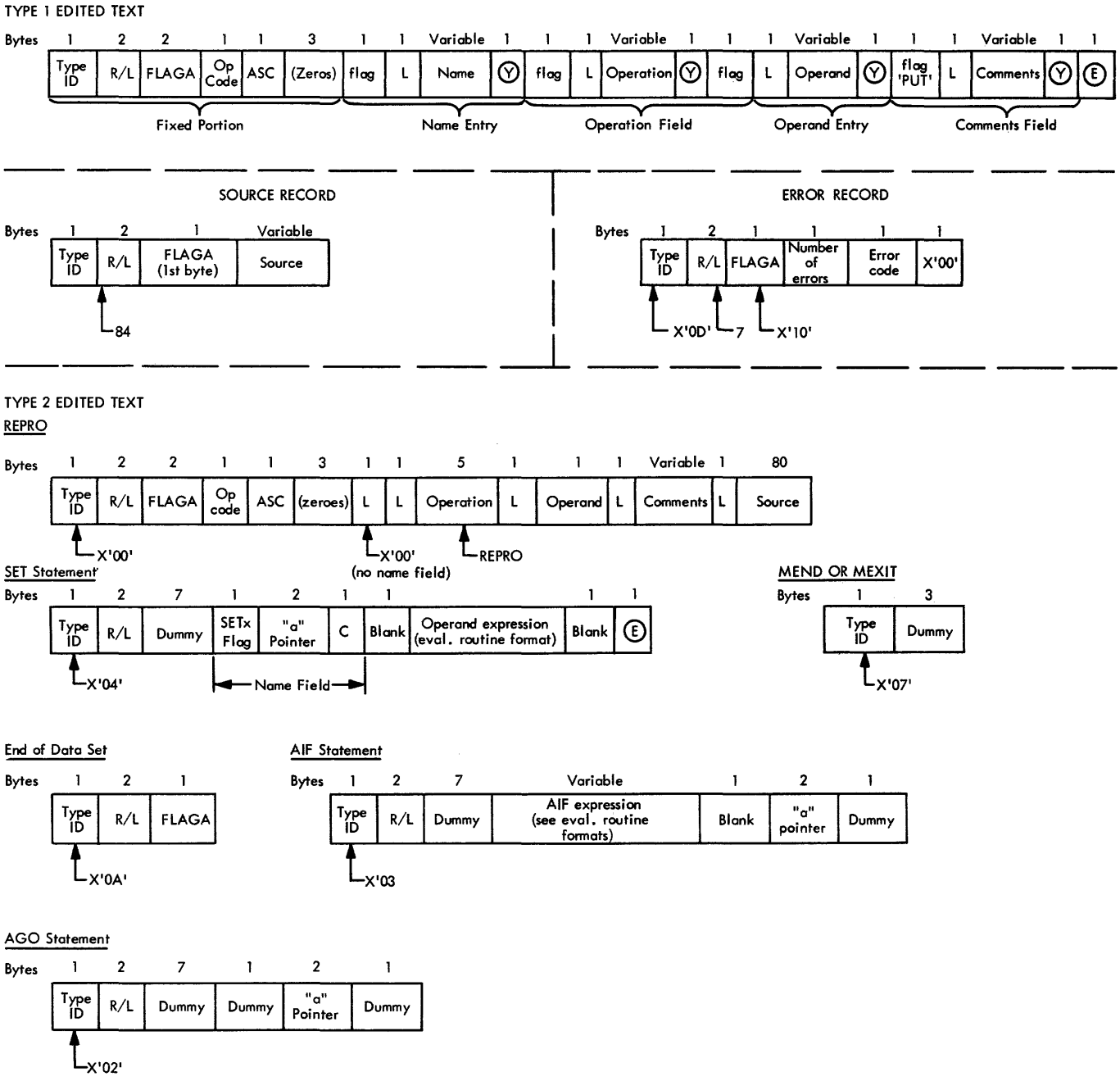



Figure B3. Macro Generator Record Formats (1 of 2)

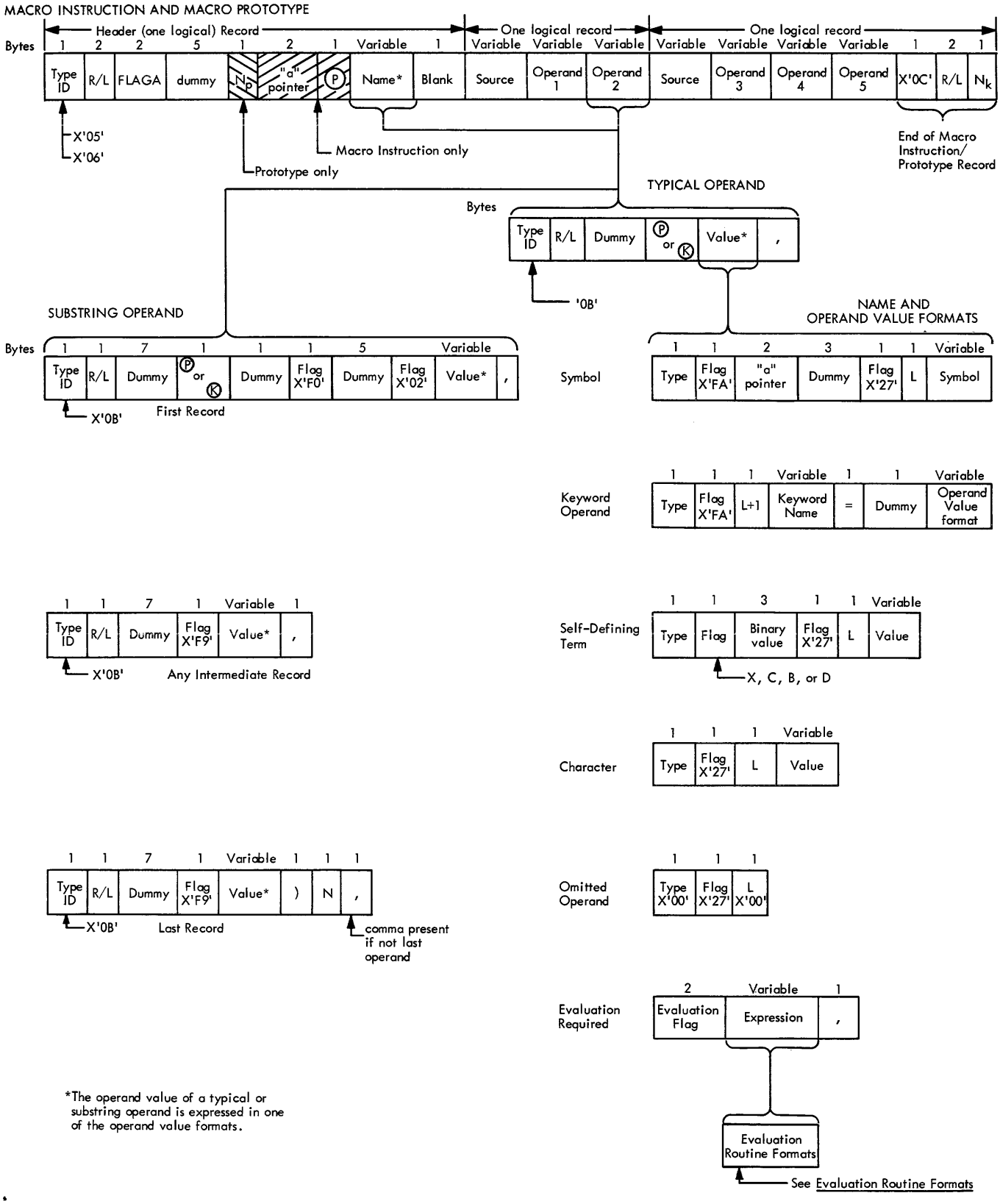



Figure B3. Macro Generator Record Formats (2 of 2)

Table B2. Assembler Operation Codes

| Mnemonic | Hexadecimal Value |
|---|---|
| GBLA | 0 |
| GBLB | 1 |
| GBLC | 2 |
| LCLA | 3 |
| LCLB | 4 |
| LCLC | 5 |
| SETA | 6 |
| SETB | 7 |
| SETC | 8 |
| AIF | 9 |
| AGO | A |
| ANULL | B |
| COPY | C |
| MACRO | D |
| MNOTE | E |
| MEXIT | F |
| MEND | 10 |
| ICTL | 11 |
| ISEQ | 12 |
| PRINT | 13 |
| SPACE | 14 |
| EJECT | 15 |
| PUNCH | 16 |
| REPRO | 17 |
| TITLE | 18 |
| ENTRY | 19 |
| EXTRN | 1A |
| START | 1B |
| CSECT | 1C |
| DSECT | 1D |
| COM | 1E |
| EQU | 1F |
| ORG | 20 |
| END | 21 |
| LTORG | 22 |
| USING | 23 |
| DROP | 24 |
| Literal DC | 25 |
| DC | 26 |
| DS | 27 |
| CCW | 28 |
| CNOP | 29 |
| DXD | 2B |
| CXD | 2C |

phases. See Edited Text Record Fixed Field Format (TXASC) in Phase F7.)

Flag - See Table B3. The flag byte indicates such things as logical operators, self-defining term type, symbol attribute type, macro parameter type, etc. The flag bytes in the record format illustrations are normally identified by their actual hexadecimal value or by a circle symbolic representation (such as Ⓨ ).

L - The length of the field which follows this byte. If L is zero, the field which normally follows it is not present. L preceeds the name, operation, operand, and comments fields in edited text and the value field in Operand Value Formats.

Type - Internal value for type attribute. See Table B4.

$N_p$ - Number of positional operands in a macro prototype.

$N_k$ - Number of keyword operands in a macro instruction or prototype.

$N_s$ - Number of elements in a sublist operand.

Dummy - Unused bytes.

"a" pointer - Little "a" pointer to a location in the subsetted dictionary of the entry for this mnemonic.

C - Bit 0 Dictionary bit

0 - Local
1 - Global

5-7 SETB bits

Table B3. Flag Values

| 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 |
| 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 | &SYSLIST |
| F0 | Sublist |

Table B3. Flag Values (Continued)

| Value (Hex.) | Flag Description |
|---|---|
| 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 B4. 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' |

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.

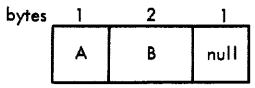
Operand Reference

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

| | |
|---|---|
| 0 | $SYSNDX |
| 1 | $SYSECT |
| 2 | Symbolic parameter in name field of prototype statement |
| 3 ⋮ 202 | Operand field of prototype statement |

Keyword operands are given a position number similar to positional operands. The positions are assigned in the order that they appear in the operand field of the prototype statement.

Format of request for substitution of macro instruction operand in place of symbolic parameter:

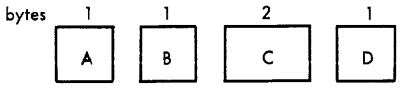



A - Operand request flag (X'39')
B - Operand number

Evaluation Routine Formats

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

Attributes (L'I'S'T')



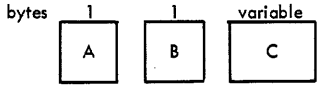

A - Flag byte (type of attribute). See Table B3.

B - Symbolic parameter (X'39') or symbol flag (X'FA'). See Table B3.

C - 2-byte "a" pointer if the reference is to an attribute of a symbol, or parameter position number in low order byte of C if the reference is to an attribute of a symbolic parameter.

D - Dummy

Character String

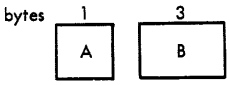



A - Flag byte 'C' (X'27')

B - True length byte (if zero, C will not exist)

C - Data bytes (variable bytes of characters)

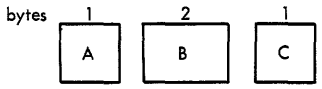
Decimal, Hexadecimal, Binary, or Character Self-Defining Term




A - Flag byte (hexadecimal = X'22', binary = X'23', decimal = X'24', character = X'25')

B - Data bytes. Three bytes of data in binary.

Variable Symbol

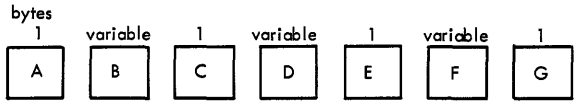



A - Flag byte (SETA = X'28' SETB = X'29', SETC = X'2A')

B - 2-byte "a" pointer

C - Bits 5-7 for subscripted SETB. Indicates in binary form the particular bit (0-7) within the byte referenced by B that contains the SETB evaulation; bit 0 = 0 for local or 1 for global.

Substring




A - Begin substring 'BEGSUB' flag (X'2C')

B - Character expression (variable bytes)

C - Begin first operand 'SUBOPE' flag (X'2D')

D - Expression 1 (variable bytes)

E - First operand completed 'SUBCOM' flag (X'2E')

F - Expression 2 (variable bytes)

G - End of substring notation 'SUBCLS' flag (X'2F')

Subscripting. Left parenthesis is replaced by a special subscript left paren flag (X'03').

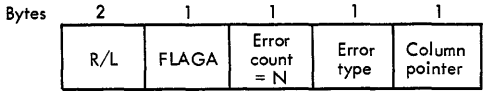
Remainder of the format is as previously described.

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

PHASE F7

Record Formats

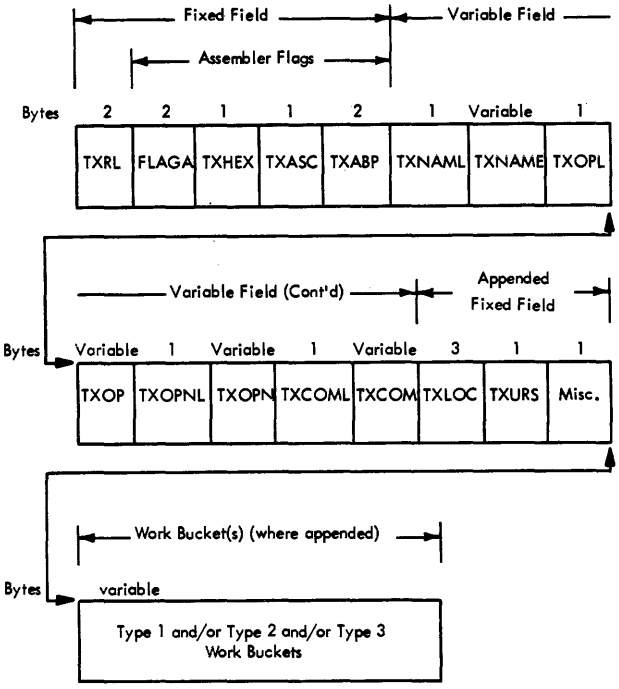
Error Record




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 edited text records in a format prepared by the macro generator. Phase F7 processes these records and attaches an "appended fixed field" and, when required, "work buckets". The description of the F7 edited text records that follows gives the field names as they appear in the listings.



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.

FLAG A -

Bit 0 TXLRI. 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 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.

Bit 0 0 - No floating point register required.
1 - Floating point register required.

Bit 1 0 - No even register required.
1 - Even register required (0, 2, 4, 6); or register 0 or 4 required.

Bits 2-3 00 - No boundary alignment.
01 - Half word.
10 - Full word.
11 - Double word boundary alignment.

Bits 4-5 Type of class within instruction (XX + 1).

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 B4.

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.

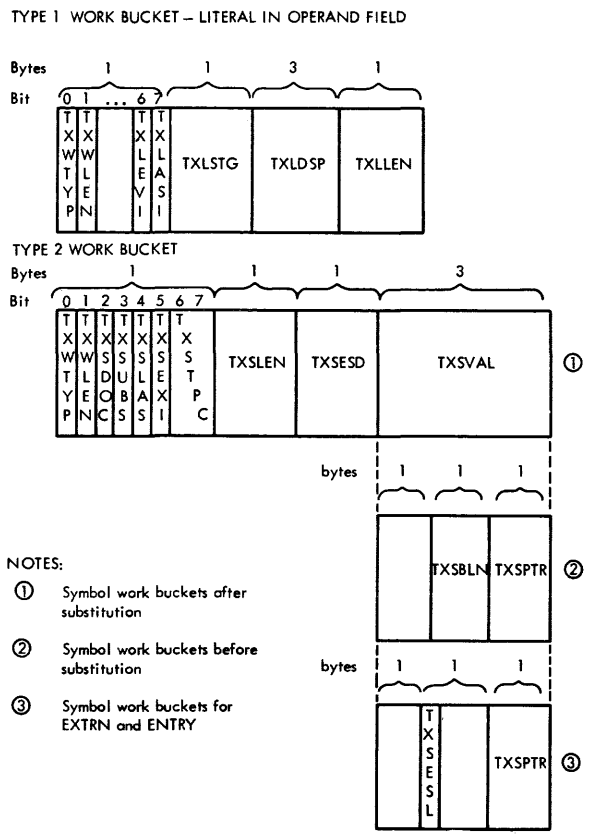



Figure B4. Types 1 and 2 Work Buckets

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

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 B4. 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 B5. 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).

Bit 2 TXDPPI. DC previously processed indicator.

Bit 3 TXDLMP. Length modifier present indicator.

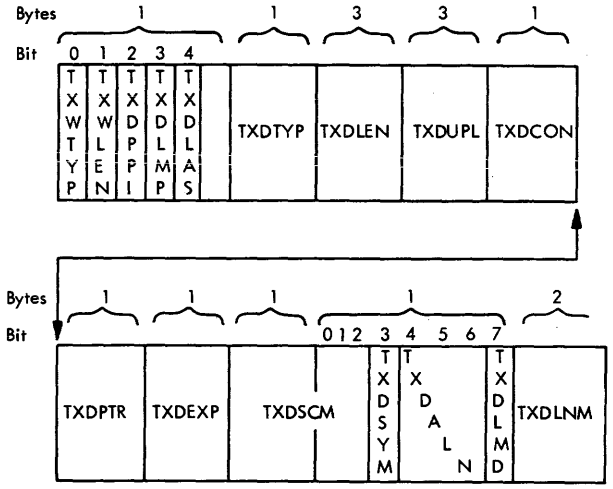



Figure B5. Type 3 Work Bucket

Bit 4 TXDLAS. Last operand indicator.

Bits 5-7 (Blank).

TXDTYP - Type, translated. See Table B5.

Table B5. 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 |
| 0A | Y CON |
| 0B | V-CON |
| 0C | S-CON |
| | Address Constants |

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.

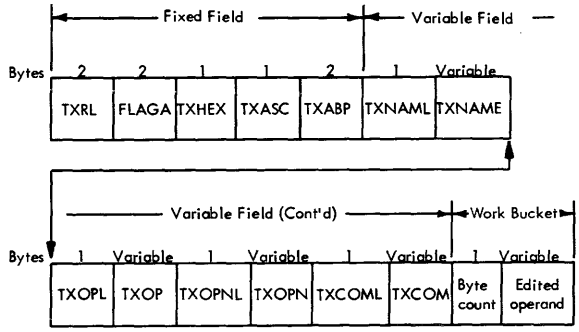
TXDALN - Alignment

TXDLMD - Length modifier type.

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

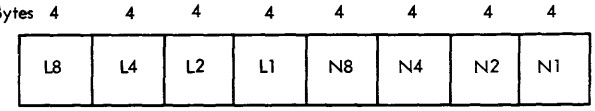



The eight-byte fixed field is the same as that described under "Edited Text Record Fixed Field Format." The variable field is the same as that described under "Edited Text Record Variable Field Format." However, in place of an appended fixed field is a special work bucket, as follows:

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 Work Bucket.



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.

N2 - Number of entries in 2-byte chain.

N1 - Number of entries in 1-byte chain.

## Tables

### Symbol Table

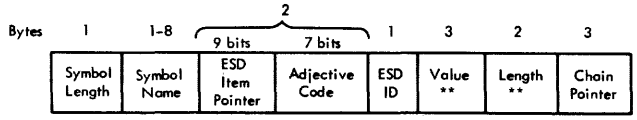
The symbol table is a collection of symbols and literals with their associated attributes. It is built during Phase F7.

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 two types of entries in the symbol table.

1. Name entries.
2. Literal entries.

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




Adjective code -

Bit 1 Not used.

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

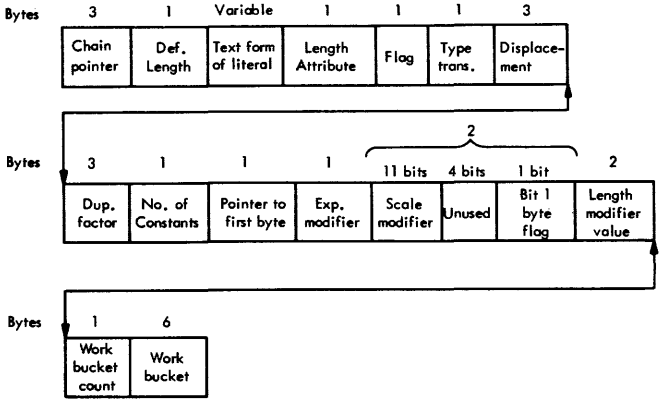
00 - CSECT
01 - EXTRN
10 - DSECT
11 - NAME

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.

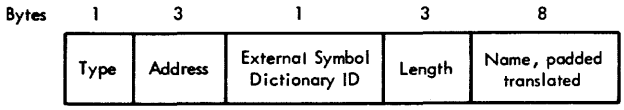
Literal Entries.



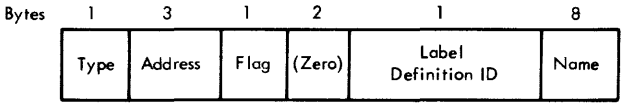

### External Symbol Dictionary

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

Control Sections (CSECT) and External References (EXTRN).




Entry Definitions.




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).

| Bytes 1 | 2 | 1 | 1 | 3 | 8 |
|---|---|---|---|---|---|
| Type | ESDNO | Alignment | ESDID | Length | Name |

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 Entries

| 1 | 3 | 3 | 3 | 3 |
|---|---|---|---|---|
| ESDID | Location | Length (8-byte string) | Length (4-byte string) | Length (2-byte string) |

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.

Cross Reference Dictionary Entries

| 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

PHASE FI

Literal Adjustment Table

| 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 that immediately follow this byte.

A - The adjusted assembler address of the beginning of the 8-byte string of literals whose pool is described by this table.

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

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

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 |

PHASE F8

Relocation Dictionary Entries

| 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'.

Position 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-, Y-, and Q-type address constants.
- 01 - V-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.

| Macro Instruction | |
|---|---|
| GET | Reads logical records from files organized by the file definition macro instruction. |
| PUT | Writes logical records into files organized by the file definition macro instruction. |
| READ | Reads the next sequential physical record from a file organized by the file definition macro DTFMT (define the file for magnetic tape). |
| WRITE | Writes a physical record or a portion of a physical record onto a file organized by the file definition macro DTFMT. |

NOTE: DOS with TWFs uses DTFMT and DOS with DWFs uses DTFSD.

| | |
|---|---|
| CHECK | Waits (if necessary) for the completion of a READ or WRITE operation and detects errors and exceptional conditions. |
| NOTE | Obtains the relative position of the last physical record that was read or written from a specified file. |

| Macro Instruction | |
|---|---|
| POINTR | Repositions a file so that the next read operation involves a record previously identified by a NOTE macro instruction. |
| POINTW | Repositions a file so that the next write operation involves a record previously identified by a NOTE macro instruction. |
| POINTS | Repositions a file to the first record. |
| OPEN | Makes a file available for use. |
| CLOSE | Makes a file unavailable for use. |
| FETCH | Loads and transfers control to another phase of the assembler. |
| LOAD | Loads a phase or program segment and returns control to the caller. |
| CNTRL | Performs certain physical, non-data operations on the device associated with the specified file, e.g., rewind file. |

## APPENDIX D. ASSEMBLER ORGANIZATION

The physical organization of the assembler differs somewhat from the logical organization described in this manual. This appendix summarizes the physical organization of the assembler and correlates it with the logical organization.

The first eight bytes of each core image phase is the phase identifier. It indicates the phase, version number, and level number of the assembler.

Table D1 is an annotated Linkage Editor map. It was produced by a DOS system with a 10K supervisor. Note that the term "phase", as used throughout the manual, means "logical phase". To DOS however, a phase is a unit of code -- consisting of one or more Relocatable Library modules -- which exists in the Core Image Library. The Linkage Editor map shows the core image phases and CSECTS which make up the assembler, the overlay structure, and the relationship of logical phases and physical structure.

Table D2 is a storage allocation map which shows the relative location of the CSECTs, dictionaries and tables, and buffers. It also illustrates the overlay structure of the assembler.

Table D1. Annotated Linkage Editor Map

```
JOB  LINK      12/14/67   DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN    MAP
LIST   INCLUDE IJYASM
LIST           PHASE     ASSEMBLY,S,NOAUTO
LIST           INCLUDE   IJYFO,(IJYFORTO,IJYFOMTM,IJYFOSDM,IJYFOGIO)
LIST           INCLUDE   IJYCM
LIST           INCLUDE   IJYIN
LIST           INCLUDE   IJYF2
LIST           INCLUDE   IJYF1
LIST           INCLUDE   IJYFO,(IJYFOTDF)
LIST           PHASE     ASSEM3,IJYCMORG,NOAUTO
LIST           INCLUDE   IJYF3
LIST           PHASE     ASSEM3E,IJYCMORG,NOAUTO
LIST           INCLUDE   IJYF3E
LIST           PHASE     ASSEM7,IJYFOGIO,NOAUTO
LIST           INCLUDE   IJYRTA
LIST           INCLUDE   IJYF7I
LIST           INCLUDE   IJYF7E
LIST           INCLUDE   IJYF7D
LIST           INCLUDE   IJYF7X
LIST           INCLUDE   IJYF7N
LIST           INCLUDE   IJYF7V
LIST           INCLUDE   IJYF7L
LIST           INCLUDE   IJYF7G
LIST           INCLUDE   IJYF7C
LIST           INCLUDE   IJYF7S
LIST           PHASE     ASSEMFI,IJYRTA04,NOAUTO
LIST           INCLUDE   IJYRTB
LIST           INCLUDE   IJYFIO
LIST           PHASE     ASSEMF8,IJYFIO,NOAUTO
LIST           INCLUDE   IJYF8I
LIST           INCLUDE   IJYF8C
LIST           INCLUDE   IJYF8M
LIST           INCLUDE   IJYF8A
LIST           INCLUDE   IJYF8P
LIST           INCLUDE   IJYF8D
LIST           INCLUDE   IJYF8V
LIST           INCLUDE   IJYF8S
LIST           INCLUDE   IJYF8L
LIST           INCLUDE   IJYF8N
LIST           PHASE     ASSEMFPP,IJYFIO,NOAUTO
LIST           INCLUDE   IJYFPP
LIST           INCLUDE   IJYFD
LIST           PHASE     ASSEMABT,IJYFIO,NOAUTO
LIST           INCLUDE   IJYABT
LIST           ENTRY     IJYF1BGN
```

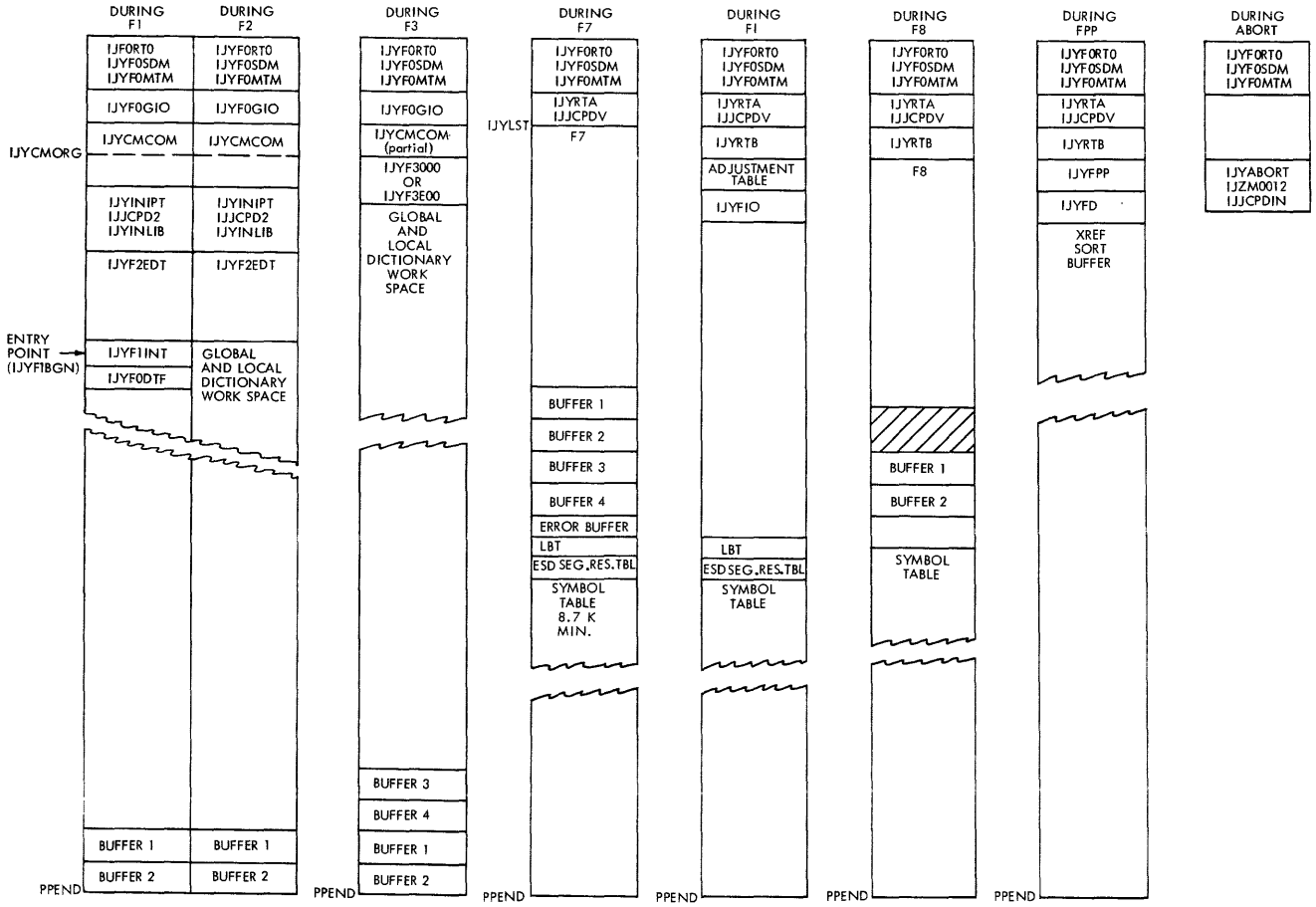
Table D1. Annotated Linkage Editor Map

| Logical Phase | PHASE | XFR-AD | LOCORE | HICORE | DSK-AD | ESD TYPE | LABEL | LOADED | REL-FR | CSECT Description |
|---|---|---|---|---|---|---|---|---|---|---|
| F0 | ASSEMBLY | 0085E0 | 002800 | 0093D3 | 1C 7 2 | CSECT | IJYF0RT0 | 002800 | 002800 | Master Root Segment -- contains utility DTF's (DTFSD) and linkage to phase ABT |
| | | | | | | ENTRY | IJYF0S01 | 002820 | | |
| | | | | | | ENTRY | IJYF0S02 | 0028B8 | | |
| | | | | | | ENTRY | IJYF0S03 | 002950 | | |
| | | | | | | ENTRY | IJYF0ESW | 00281C | | |
| | | | | | | * ENTRY | IJYF0ABT | 002808 | | |
| | | | | | | CSECT | IJYF0SDM | 002C08 | 002800 | Disk work file I/O logic module (SDMODW) |
| | | | | | | CSECT | IJYF0MTM | 0029E8 | 002800 | Tape work file I/O logic module (MTMOD) |
| | | | | | | CSECT | IJYF0GIO | 002F88 | 002800 | Macro Generator I/O interface routines |
| | | | | | | ENTRY | IJYF0CHK | 002FE0 | | |
| | | | | | | ENTRY | IJYF0NTE | 003092 | | |
| | | | | | | ENTRY | IJYF0PTR | 003040 | | |
| | | | | | | ENTRY | IJYF0PTW | 003004 | | |
| | | | | | | ENTRY | IJYF0RED | 002FB4 | | |
| | | | | | | ENTRY | IJYF0WRT | 002F88 | | |
| | | | | | | CSECT | IJYF0TDF | 009320 | 008A48 | Alternate tape work file DTF's (DTFMT) |
| FCOM | | | | | | CSECT | IJYCMCOM | 0030D8 | 0030D8 | Macro Generator Common -- contains common constants and communications work area |
| | | | | | | * ENTRY | IJYCMORG | 003130 | | |
| FIN | | | | | | CSECT | IJYINIPT | 003C18 | 003C18 | SYSIPT DTF (DTFCP) |
| | | | | | | ENTRY | IJYINBF1 | 003C94 | | |
| | | | | | | CSECT | IJJCPD2 | 003D38 | 003C18 | SYSIPT I/O logic module (CPMOD) |
| | | | | | | ENTRY | IJJCPD3 | 003D38 | | |
| | | | | | | CSECT | IJYINLIB | 003E38 | 003C18 | SYSSLB I/O logic (DTFSL) |
| F2 | | | | | | CSECT | IJYF2EDT | 004178 | 004178 | Input and macro editing |
| | | | | | | ENTRY | IJYF2GLD | 0079D8 | | |
| | | | | | | ENTRY | IJYF2EDC | 006148 | | |
| | | | | | | ENTRY | IJYF2DCL | 0072D0 | | |
| | | | | | | ENTRY | IJYF2EOF | 0077E6 | | |
| | | | | | | ENTRY | IJYF2GSC | 00757E | | |
| F1 | | | | | | CSECT | IJYF1INT | 0085D8 | 0085D8 | Initialization |
| | | | | | | * ENTRY | IJYF1BGN | 0085E0 | | |
| F3 | ASSEM3 | 003138 | 003130 | 004F9F | 1D 5 2 | CSECT | IJYF300U | 003130 | 003130 | Macro generation and conditional assembly |
| F3E | ASSEM3E | 003138 | 003130 | 00349F | 1D 8 1 | CSECT | IJYF3E00 | 003130 | 003130 | Macro Generator abort phase |
| RTA | ASSEM7 | 004552 | 002F88 | 008C4F | 1D 8 2 | CSECT | IJYRTA | 002F88 | 002F88 | ACT equates and translate table |
| | | | | | | CSECT | IJJCPDV | 003090 | 002F88 | SYSLNK, SYSLST, and SYSPCH logic module (CPMOD); SYSPCH and SYSLNK DTF's (DTFCP);utility file I/O routines; Assembler Control Table (ACT) |
| | | | | | | ENTRY | IJYRTA01 | 0032A0 | | |
| | | | | | | ENTRY | IJYRTA02 | 003388 | | |
| | | | | | | * ENTRY | IJJCPDV1 | 003090 | | |
| | | | | | | ENTRY | IJJCPD0 | 003090 | | |
| | | | | | | ENTRY | IJJCPDON | 003090 | | |
| | | | | | | ENTRY | IJYRTA03 | 003420 | | |
| | | | | | | * ENTRY | IJYRTA04 | 004078 | | |
| F7 | | | | | | CSECT | IJYF7I | 004078 | 004078 | I/O subroutines |
| | | | | | | CSECT | IJYF7E | 004C80 | 004C80 | External Symbol Dictionary (ESD) processor |
| | | | | | | ENTRY | IJYF7E02 | 004CAC | | |
| | | | | | | ENTRY | IJYF7E04 | 004D76 | | |
| | | | | | | ENTRY | IJYF7E05 | 004E48 | | |
| | | | | | | ENTRY | IJYF7E07 | 004F3A | | |
| | | | | | | ENTRY | IJYF7E08 | 004FCE | | |
| | | | | | | ENTRY | IJYF7E06 | 004F0E | | |
| | | | | | | ENTRY | IJYF7E09 | 005070 | | |
| | | | | | | ENTRY | IJYF7E10 | 0050FA | | |
| | | | | | | ENTRY | IJYF7E11 | 005158 | | |
| | | | | | | ENTRY | IJYF7E12 | 0051E8 | | |
| | | | | | | ENTRY | IJYF7E03 | 004D04 | | |
| | | | | | | ENTRY | IJYF7E01 | 004C80 | | |
| | | | | | | ENTRY | IJYF7E13 | 005238 | | |
| | | | | | | CSECT | IJYF7D | 005628 | 005628 | DC/DS evaluation routine |
| | | | | | | CSECT | IJYF7X | 006498 | 006498 | GET statement |
| | | | | | | CSECT | IJYF7N | 006EE8 | 006EE8 | AUTOTEST processor |
| | | | | | | CSECT | IJYF7S | 008910 | 008910 | Symbol Table Processor |
| | | | | | | ENTRY | IJYF7SND | 008C50 | | |
| | | | | | | ENTRY | IJYF7S01 | 008910 | | |
| | | | | | | ENTRY | IJYF7S02 | 008916 | | |
| | | | | | | ENTRY | IJYF7S03 | 008AEC | | |

Table D1. Annotated Linkage Editor Map

| Logical Phase | PHASE | XFR-AD | LOCORE | HICORE | DSK-AD | ESD TYPE | LABEL | LOADED | REL-FR | CSECT Description |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | CSECT | IJYF7V | 0071B8 | 0071B8 | Expression evaluation routine |
| | | | | | | CSECT | IJYF7L | 007A68 | 007A68 | Error logging routine |
| F7 | | | | | | CSECT | IJYF7G | 007B38 | 007B38 | Literal DC generator |
| | | | | | | CSECT | IJYF7C | 007C30 | 007C30 | Mainline control routine |
| | ASSEMFI | 0047A8 | 004078 | 006ADF | 1E 5 2 | CSECT | IJYRTB | 004078 | 004078 | RTB -- contains SYSLST DTF (DTFCP) and output routine |
| | | | | | | ENTRY | IJYRTB01 | 0041D0 | | |
| FI | | | | | | ENTRY | IJYRTB02 | 00435C | | |
| | | | | | | CSECT | IJYFI0 | 0047A0 | 0047A0 | ESD writing routine |
| | ASSEMF8 | 004B94 | 0047A0 | 009393 | 1E 9 1 | CSECT | IJYF8I | 0047A0 | 0047A0 | Initialization and I/O routines |
| | | | | | | CSECT | IJYF8C | 004D78 | 004D78 | Mainline control routine |
| | | | | | | ENTRY | IJYF8C01 | 004FA4 | | |
| | | | | | | CSECT | IJYF8M | 005140 | 005140 | Machine operation processor |
| | | | | | | ENTRY | IJYF8M01 | 005748 | | |
| | | | | | | ENTRY | IJYF8M02 | 005926 | | |
| | | | | | | CSECT | IJYF8A | 005DE8 | 005DE8 | Assembler operation processor |
| | | | | | | ENTRY | IJYF8A01 | 006658 | | |
| | | | | | | CSECT | IJYF8P | 006978 | 006978 | Listing and object deck output routine |
| | | | | | | ENTRY | IJYF8P03 | 007150 | | |
| | | | | | | ENTRY | IJYF8P02 | 006FBE | | |
| F8 | | | | | | ENTRY | IJYF8P04 | 0073AF | | |
| | | | | | | ENTRY | IJYF8P01 | 006C12 | | |
| | | | | | | CSECT | IJYF8D | 007758 | 007758 | DC evaluation routine |
| | | | | | | ENTRY | IJYF8D01 | 00814E | | |
| | | | | | | ENTRY | IJYF8D02 | 0082BC | | |
| | | | | | | CSECT | IJYF8S | 008CD0 | 008CD0 | Symbol Table subroutine |
| | | | | | | CSECT | IJYF8V | 008458 | 008458 | Expression evaluation subroutine |
| | | | | | | CSECT | IJYF8L | 008D90 | 008D90 | Log error subroutine |
| | | | | | | CSECT | IJYF8N | 008E60 | 008E60 | Floating/fixed-point conversion routine |
| | | | | | | CSECT | IJYF8P05 | 0075C8 | 006978 | Output routine constants |
| | ASSEMFPP | 0047A8 | 0047A0 | 009A1F | 1F 5 1 | CSECT | IJYFPP | 0047A0 | 0047A0 | Cross-Reference and RLD processor |
| FPP | | | | | | CSECT | IJYFD | 008270 | 008270 | Error record processor |
| | | | | | | ENTRY | IJYFDEND | 009A20 | | |
| | ASSEMABT | 0047A8 | 0047A0 | 004DC6 | 20 1 2 | CSECT | IJYABORT | 0047A0 | 0047A0 | Assembly abort routine and Syslog DTF (DTFCN) |
| ABORT | | | | | | CSECT | IJZM0012 | 004C20 | 0047A0 | DTFCN logic portion |
| | | | | | | CSECT | IJJCPD1N | 004C40 | 0047A0 | SYSLST I/O logic module (CPMOD) |

## Table D2. Storage Allocation Map




NOTES:

3625 Bytes -- Max. Buffer Size

64K Bytes -- Max. Dictionary and Table Sizes

PPEND -- Upper limit of core available to Assembler

## 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 pointers) which point to locations in the dictionaries/tables. The range of the hash table is the number of such pointers that can be placed in the 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 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 procedure 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 E1.

### Backward Chaining Techniques

The record to be entered is hashed to an index number. This index number is used

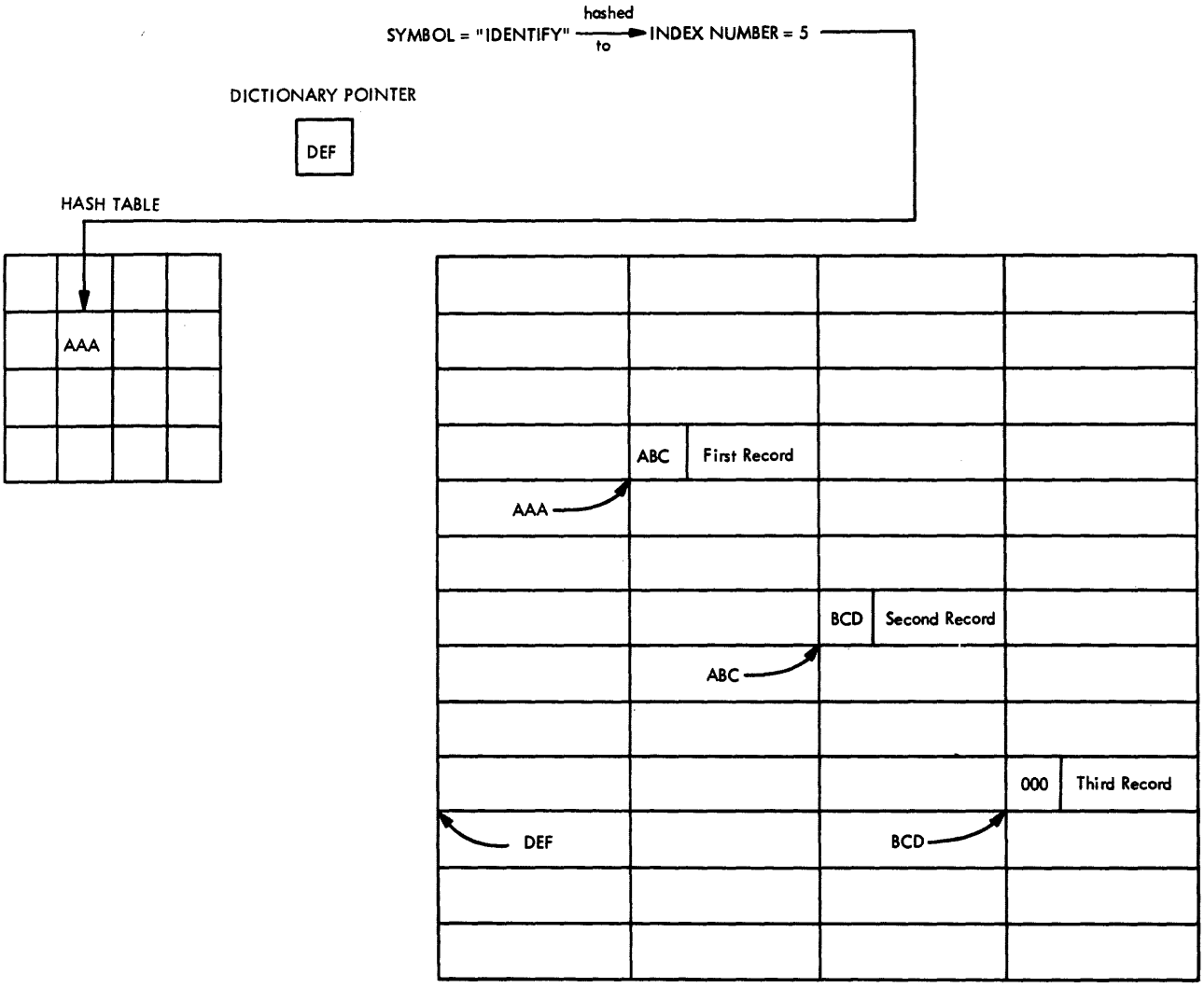



Figure E1. Hash Table and Forward Chaining

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 E2.

## 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 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 built 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.

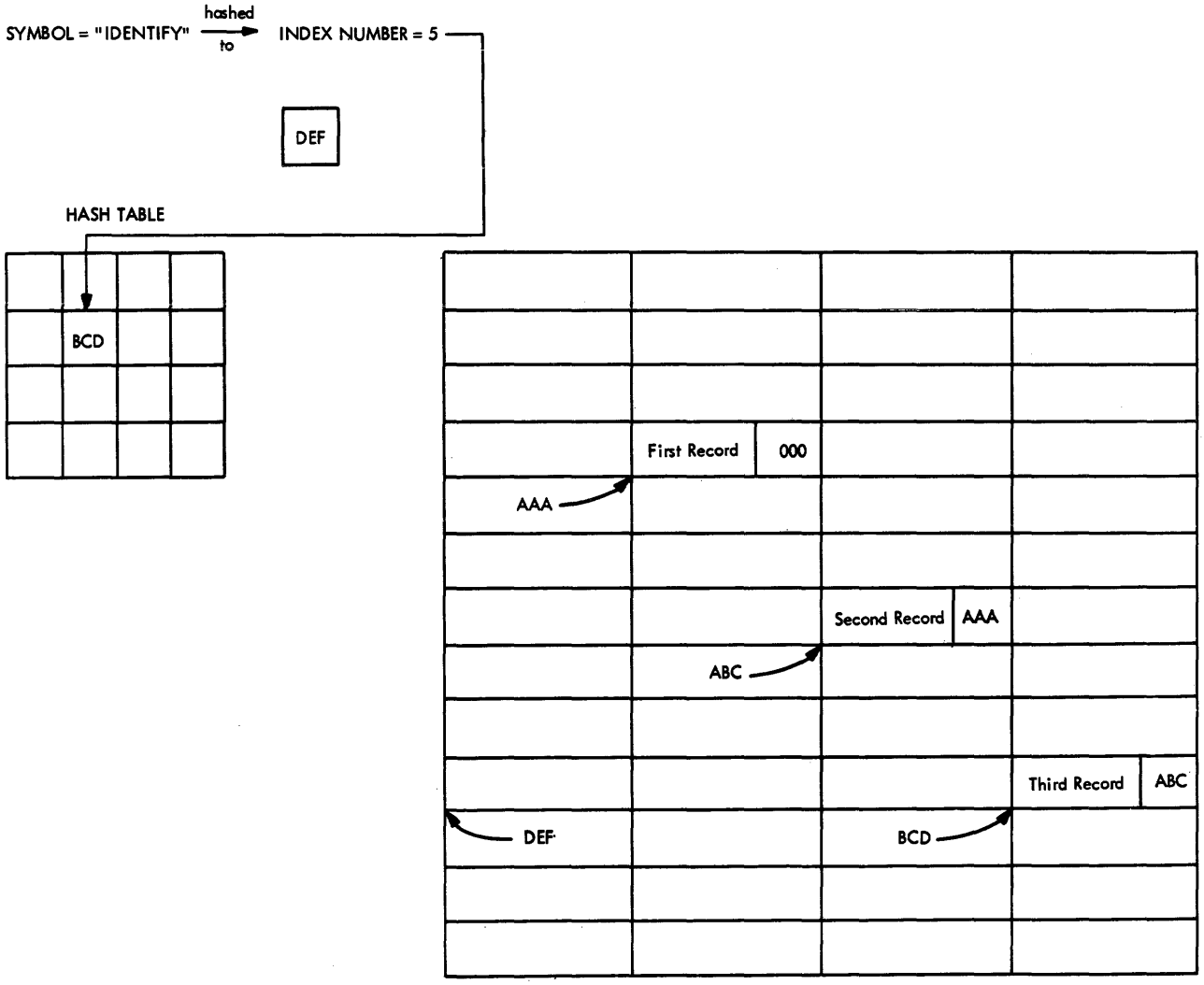



Figure E2. Hash Table and Backward Chaining

## APPENDIX F. INTERNAL ASSEMBLER CODE 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 DOS/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, below.

Application of the translate table also allows the user to assemble programs written in other than DOS/360 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.

| 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 | | |

These switches are set in the code of the macro generator and assembler phases. They do not appear in any dictionaries, tables, or records.

PHASE F3 SWITCHES

MISWIT

| Bit No. | Hex. Sw. | |
|---|---|---|
| 0 | X'80' | Not used |
| 1 | X'40' | 0 - Do another pass on macro instruction operands<br>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 is to be made from prototype<br>1 - Entry to be made from macro instrucion |
| 5 | X'04' | 1 - Nest aborted; no room to store |
| 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<br>1 - No need to NOTE in IOMAC routine |
| 4 | X'08' | 0 - Use PNTR in IOMAC routine<br>1 - Do logical POINTW 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 - SYSLST to provide alternate to symbolic parameters |
| 1 | X'40' | 1 - 2 expressions in SYSLST |
| 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<br>1 - character expression 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 | F8INST | 16 Byte Instruction Bldg. Area |
| CTSDVI | Self Defining Value Indicator | F8ZRO | One Full Word Of Zero |
| CTLCRI | Location Counter Reference Indicator | PYRSW | |
| CTMODE | Mode Indicator | F8YDC | |
| CBDNO | Blank DSECT ESD No | CTESRN | ESD Seg.Count |
| CBDSW | Blank DSECT ID No | | |

# APPENDIX H. GLOSSARY

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.

Concatenation: 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.

Core Image Module: An executable logical unit of coding. It is the output of the linkage editor in a format suitable for loading into main storage.

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.

DTF (Define The File) Table: 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.

Edited Text: Source text with appended work and code fields which map and describe its attributes.

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.

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.

Open Code: All source statements except those generated from macro definitions. Open code is read from SYSIPT.

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 source language that is input to the translation process.

Subsetted Global Dictionary: The global dictionary -- passed to Phase F3 -- which contains only global variable symbols. The machine and assembler operation codes and the macro names are no longer needed and have been removed.

Subsetted Local Dictionary: A local dictionary -- passed to Phase F3 -- after it has been sorted on the little "a" pointer and the symbols, big "A" pointer, and little "a" pointer have been removed.

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.

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

Utility Data Set: 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.

## READER'S COMMENT FORM

IBM System/360 Disk Operating System Assembler [F] Form Y26-3716-0

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

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

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

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

Other _____

- Please give specific page and line references with your comments when appropriate.

## COMMENTS

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

## YOUR COMMENTS, PLEASE...

This SRL bulletin is one of a series which serves as reference sources 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. 2078
SAN JOSE, CALIF.

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

POSTAGE WILL BE PAID BY . . .

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

Attention: Programming Publications, Dept. 232

fold fold

IBM S/360 Printed in U.S.A. Y26-3716-0

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]

IBM S/360 Printed in U.S.A. Y26-3716-0

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]