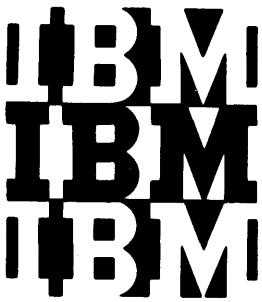


Z26-3752-0

Program Logic

IBM 1130 MONITOR PROGRAMMING SYSTEM PROGRAM LOGIC MANUAL

This publication describes the internal logic of the IBM 1130 Monitor Programming System. The contents are intended for use by persons involved in program maintenance, and for system programmers who are altering the program design. Program logic information is not necessary for the use and operation of the program; therefore, distribution of this manual is limited to those who are performing the aforementioned functions.

PREFACE

Effective use of this publication requires an understanding of the IBM 1130 Computing System and the appropriate programming system. Publications relating to the 1130 System are listed in the IBM 1130 Bibliography (Form A26-5916).

The contents of this publication describe the internal structure of the programs comprising the 1130 Disk Monitor System. The publication is divided into seven sections, the first of which is an introduction. Following this are sections which describe each of the Monitor programs:

- Supervisor
- Disk Utility Program
- Assembler Program
- FORTRAN Compiler
- Subroutine Library
- System Loader

Each section consists of a general description of the program and includes flowcharts depicting the major program components. The component names are the same as those used in the program listings supplied by the Programming Systems Department.

It has been necessary to define many terms in order to describe the 1130 Disk Monitor System. Included in this publication is a glossary of special terms. It is recommended that the reader familiarize himself with these terms before attempting to read the rest of the publication.

As an aid to the reader, actual core addresses have been specified in the text and on core maps; actual disk sector addresses have also been specified where useful. These addresses are accurate at the time of publication of this manual. However, they are subject to change and must not be construed as applicable at all times and in all cases.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. A form has been provided at the back of this publication for reader's comments. If the form has been detached, comments may be directed to: IBM, Programming Publications Dept. 452, San Jose, Calif. 95114

CONTENTS

ILLUSTRATIONS

The major characteristic of a non-process control monitor is that it allows continuous operation of stacked input jobs. It operates in a static environment in that control is regulated by the input data, not by external stimuli. It differs from a process control monitor in that the supervisory section relinquishes complete control to the program whose operation has been requested.

In the 1130 Disk System Monitor, the JOB control record defines the starting and ending points of the job; however, the total job can consists of many subjobs. The Assembler Program, the FORTRAN compiler, the Disk Utility Program, and the user's programs can be called for operation by the ASM, FOR, DUP, and XEQ control records, respectively. These are each considered subjobs, and the successful completion of the job depends on the successful completion of each subjob. In most cases all subjobs subsequent to the unsuccessful completion of a given subjob are bypassed.

The Monitor, which resides totally on disk, allows the storage and retrieval of user programs on disk by a referenced name, and it provides a work storage area on disk which can be used by both Monitor programs and user programs. A directory of programs is maintained to keep track of all programs which reside on the disk.

The overall flowchart of the Monitor is shown in Chart AA.

Supervisor

The Supervisor performs the control and loading functions of the Monitor. Monitor control records, which are used to direct the sequence of jobs without operator intervention, are included in a stacked input arrangement and are processed by the Supervisor, which decodes the control records and calls the proper Monitor program to perform the desired operation.

Disk Utility Program

The Disk Utility Program (DUP) is a group of routines that automatically allocate disk storage as required by each program stored on the disk and make these programs available in card or paper tape format, in addition to providing printed records of the status of User Storage and Working Storage. By means of DUP, the required operations of disk maintenance can be performed with minimum effort.

Assembler Program

The Assembler Program receives program source statements written in the 1130 Assembly Language and produces a machine-language program as output. The input can be in either card or paper tape format.

At the conclusion of the assembly process, the assembled program resides in Working Storage. It may also have been outputted on the principal I/O device. Assembler control records are used to specify options and to provide instructions concerning the assembly process.

FORTRAN Compiler

The FORTRAN compiler accepts statements written in the FORTRAN language as input and produces a machine-language program as output. It provides for calling the necessary subroutines at execution time.

Subroutine Library

The subroutine library consists of a group of subroutines designed to aid the programmer in making efficient use of the machine system. The library contains input/output, data conversion, arithmetic, functional, and selective dump subroutines. The user can delete undesired subroutines from the library as well as add subroutines of his own.

System Loader

The System Loader is a program that must be used initially to load the Monitor onto the disk pack. The Monitor is supplied to the user on cards or paper tape, which, with the aid of some control records (IBM-supplied in the case of Paper Tape Systems, user-supplied in the case of Card Systems), must be loaded to the disk pack before operation of the Monitor can begin.

INTRODUCTION

The Supervisor performs the control and loading functions for the 1130 Disk Monitor System. In order to accomplish the control functions, the program is divided into several segments, which are a combination of core-resident logic (Skeleton Supervisor) and separate core load phases. See Chart AB.

The relocating and loading-to-core storage functions are performed for the Supervisor by the Loader.

AREA DESCRIPTIONS

The following descriptions summarize the contents or purpose of the areas in core which contain or are used by the Supervisor. See Figure 1.

COMMA: This area is used and reserved by all Monitor programs. Among other things, it contains the Monitor indicators and switches, the LET/FLET parameters, and the Disk IOCS indicators.

Skeleton Supervisor: This area contains the Skeleton Supervisor segment of the Supervisor. The Skeleton Supervisor is located in this area except during the execution of either a FORTRAN core load or an Assembler core load which uses the DISKZ routine.

I/O Area (low-core): This area holds the I/O routines (Card or Paper Tape, Disk, and Console Printer) that are used by the Supervisor. The zero, e.g., CARD0, versions of the I/O routines are used by the Supervisor (except for paper tape, in which case it uses PAPT1.)

This area is overlaid by the user Disk I/O and/or the user's core load at load time.

Presupervisor: This area may also be overlaid by the user's core load. It contains the Presupervisor segment of the Supervisor and a routine which gives DISK0 the ability to process multiple sectors.

Phase Area: This area contains the phases of the Supervisor during its execution.

I/O Area (high-core): This area contains the character code conversion routines, the 1132 Printer routine, and the I/O buffer. This area may also be overlaid by the user's core load at load time.

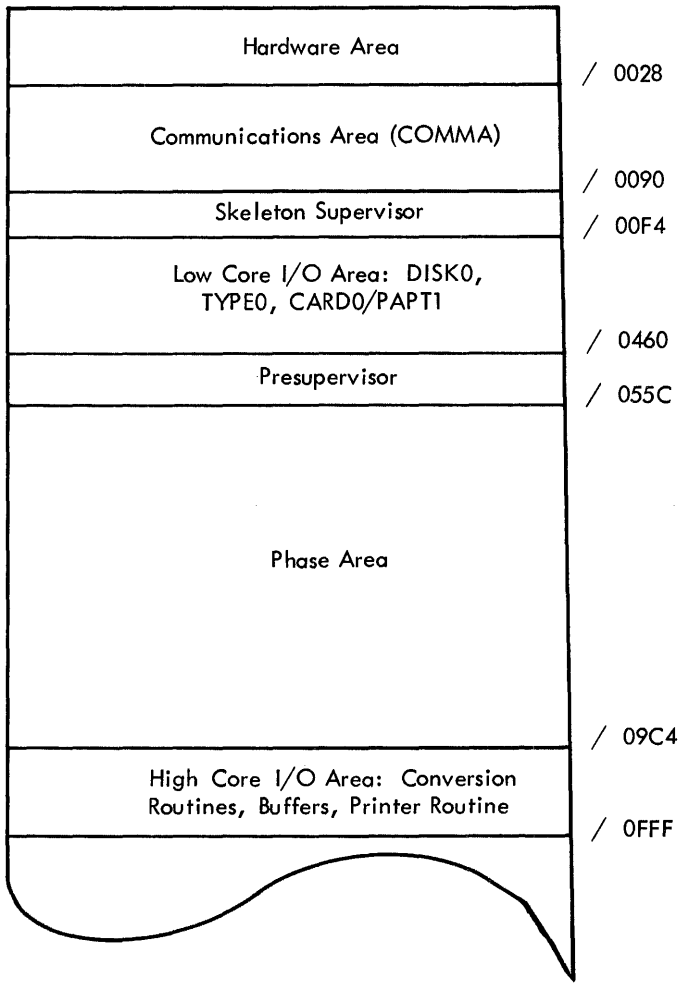


Figure 1. Supervisor Phases and Areas

CONTROL FUNCTIONS

RESIDENT ROUTINES

Skeleton Supervisor

Chart: AC

- Calls the Presupervisor on CALL EXIT or CALL LINK.
- Completes the loading of the user's core load and transfers control to that program.

Upon a CALL EXIT or a CALL LINK from a successfully completed user mainline or upon the return of a Monitor program to the Supervisor, the Skeleton Supervisor calls the Presupervisor, which determines whether the return was due to a CALL EXIT or a CALL LINK.

When entered from the Loader, the Skeleton Supervisor reads the first sector of user's core load into core and transfers control to it.

When entered via CALL LINK or CALL EXIT, the Skeleton Supervisor waits for all interrupts to be serviced before continuing.

The Master IL routine, which handles all I/O interrupts during Supervisor execution, is contained within the Skeleton Supervisor ($00E0_{16}$-$00F3_{16}$).

Presupervisor

Chart: AC

- Initializes the Master IL routine.
- Saves the areas above and below the Presupervisor on the CIB.
- Loads the I/O routines used by the Supervisor.
- Calls Phase A.

For all CALL LINK entries, before reading the Supervisor into core, the Presupervisor saves core from core location 256 to core location 4095 on the CIB, except for locations 1216 - 1536, which are saved by the Skeleton Supervisor before the Presupervisor is read in.

Contained within the Presupervisor is a routine which enables DISK0 to read or write more than 320 words. Thus, the Presupervisor can use whatever disk routine happens to be in core at the time it gains control to load its own disk I/O routine.

The Presupervisor then reads DISK0 into the Disk I/O area, followed by CARD0 or PAPT1, and TYPE0.

Phase A is read into core by the Presupervisor to analyze the Monitor control record.

Input/Output Subroutines

- Perform the servicing functions for the I/O devices.

The Supervisor makes use of the I/O routines DISK0, TYPE0, and either CARD0 or PAPT1, depending upon the system configuration. These routines are used by all phases of the Supervisor. They are loaded into the low core I/O area, between the Skeleton Supervisor and the Presupervisor.

These routines are identical to the corresponding I/O routines in the Subroutine Library but are not taken from the Subroutine Library. They are wholly incorporated in the Supervisor itself.

A fourth I/O routine is used by the Supervisor to perform required printing on the 1132 Printer. This routine is loaded into the I/O area in high core.

See the Subroutine Library portion of this manual, Section 6.

I/O Conversion Routines

- Perform the I/O character code conversions.

These routines provide the interface between the internal character representation and the I/O devices. The routine EBPRT converts from EBCDIC either to Console Printer code or to hexadecimal (for the 1132 Printer). The routine HOLEB converts EBCDIC to IBM card code and vice-versa. The routine PAPEB converts EBCDIC to Paper Tape code and vice-versa.

During Supervisor execution these routines reside in the high-core I/O area. They are identical to the routines in the Subroutine Library of the same names. However, they are a part of the Supervisor and are not loaded from the Subroutine Library.

PHASES

Phase A

Chart: AD

- Initializes for the principal print device.
- Initializes for the principal I/O device.
- Reads the input record.
- Analyzes each Monitor control record and calls the requested Monitor program.

- Prints the Monitor control records.
- Calls Phase B if an XEQ record is encountered.

Phase A is the Monitor Control Record Analyzer and Director. For each type of Monitor control record, Phase A initiates and calls the requested Monitor program. In the case of an XEQ record, additional phases of the Supervisor are required to complete the processing before the Loader can be called. Phase A prints the error messages for the remaining phases of the Supervisor.

SUP1: Tests to determine the type of entry into the Skeleton Supervisor. Entry from a CALL LINK causes the program to transfer into the routine that analyzes the XEQ control record. An entry from a Cold Start or a CALL EXIT causes the program to transfer into the routine that reads the input control record.

OKSUP: Detects and prints comments record.

PRNT2: Prints the control records.

NAME 1: Places the control record name in the accumulator.

NAME 2: Tests for the type of control record; branches to JOB1, ASM1, FOR1, XEQ1, DUP1, PAUS1, TYP1, or TEND1 on a valid type; prints an error message on an invalid Monitor control record and returns to SUP1.

JOB1: Indicates the end of one job and the beginning of the next. Switches are initialized and the Monitor control record (// JOB) is printed. The control data from the Monitor // JOB record is stored and is used to modify the disk input/output control words.

PAUS1: Causes the system to enter the WAIT state to allow for operator intervention. The routine branches to SUP1 to read the next record in the stack when the PROGRAM START key is pressed.

TYP1: Causes the input mode to be switched from the principal I/O device to the keyboard for succeeding Supervisor control records. This allows the operator to type in the control records.

TEND1: Causes the input mode to be switched from the keyboard to the principal I/O device for succeeding control records.

ASM1: Causes Phase A to initialize and read the first sector of the Assembler Program into core for execution. This is the Assembler caller routine.

FOR1: Causes Phase A to initialize and read the first sector of the FORTRAN compiler into core for execution. This is the FORTRAN Compiler caller routine.

DUP1: Causes Phase A to initialize and read the first sector of the Disk Utility Program (DUP) into core for execution. This is the Disk Utility Program caller routine.

NONDP: Prevents DUP from being read if the non-DUP switch is set. A FORTRAN Compiler or Assembler diagnostic, among other things, sets the non-DUP switch. If it is set, an error message is printed and the program returns to SUP1.

XEQ1: Causes Phase A to continue the Supervisor processing by calling Phase B if either an XEQ Monitor control record is encountered or an entry to the Supervisor by a CALL LINK occurs. However, a FORTRAN Compiler or Assembler Program diagnostic sets the non-XEQ switch. If XEQ1 finds this switch set, an error message is printed and the program returns to SUP1.

Phase B

Chart: AE

- Converts the mainline name to modified EBCDIC and compresses it into name code.
- Stores the count of *LOCAL, *NOCAL, and *FILES records.
- Searches LET/FLET for the sector address of the mainline.
- Determines the format of the program: Core Image or Disk System format.
- Calls either Phase C or the Loader.

Phase B is initialized and brought into core storage when Phase A detects an XEQ Monitor control record. The information contained in the XEQ record is analyzed and processed. Phase C is called if the XEQ record indicates that *LOCAL, *NOCAL and/or *FILES

Supervisor control records follow. Otherwise, the Loader is called to load the program to be executed.

RITE: Looks up a single character in the EBCDIC table (TAB1). When the input character is verified, it is left-justified and truncated into modified EBCDIC code. VERT1 contains the converted character.

LIMIT: Determines if the converted character falls within a range of decimal 0 through 9. If so, it is right-justified and stored in VERT1. Otherwise, it is considered an invalid character.

GOXEQ: Stores each character of the mainline name in a separate word for conversion to modified EBCDIC; compresses the name into name code.

LOKUP: Performs the LET/FLET look-up. The routine NAME initializes for the LET search; FLTLK initializes for the FLET search. The equivalent address for the mainline name is obtained from either of these two tables. It is an error if the name is not located.

HIT: Analyzes the indicator bits in the located table entry to determine which Loader entry point to use for loading the program. The possible bit combinations are:

1. 00 Disk System Format Load
2. 01 An error condition: Program is considered not to be in loadable form
3. 11 An error condition: Program is considered not to be in loadable form
4. 10 Core Image Load

TST2 and TST3 routines specify that the Core Image and Disk System Format entry points, respectively, are required.

CPROC: Initializes and calls Phase C into core storage. Phase C is executed repeatedly just after the mainline name is converted from EBCDIC to name code until all of the *LOCAL, *NOCAL and/or *FILES records have been processed. Then the remainder of Phase B is executed.

Phase C

Chart: AF

- Initializes for processing *LOCAL, *NOCAL and/or *FILES records.
- Calls Phase D to process *LOCAL or *NOCAL control records and Phase E to process *FILES control records.

Phase C is initialized and brought into core storage if Phase B detects a count in the XEQ record. The count indicates the number of Supervisor control records that follow.

Phase C reads and prints the first Supervisor control record. Then, depending upon the control record type, Phase C calls either Phase D or Phase E. Phase D processes *LOCAL and *NOCAL control records; Phase E processes *FILES control records. These phases process control records until a type change is detected. All the Supervisor control records of each type MUST be processed before the type change is made. The Supervisor control record types can be processed in any order. A type change causes Phase C to be recalled. Phase C in turn calls the phase to process the new type control record.

BLNK1: Reads and prints the *-type record.

LTST: Tests for the name LOCAL in a Supervisor control record.

NTST: Tests for the name NOCAL in a Supervisor control record.

FTST: Tests for the name FILES in a Supervisor control record.

LOCAL: Initializes and calls Phase D for *LOCAL or *NOCAL processing.

FPROC: Initializes and calls Phase E for *FILES processing.

Phase D

Chart: AG

- Converts the mainline and subroutine names to name code and stores these names.
- Writes the *LOCAL and *NOCAL records on the disk.
- Reads and prints all *LOCAL or *NOCAL records (after the first).
- Calls Phase C at a type change or Phase B at the end of the Supervisor control records.

Phase D is initialized and called by Phase C to process *LOCAL and *NOCAL records. A separate pass is made for each type.

Phase D extracts from the control records the mainline and subprogram names, converts them to name code, and stores them in the disk output area. Upon detection of a control record type change, or the end of the Supervisor control records, Phase D writes the disk output area on the LOCAL/NOCAL control record area of disk.

Figure 2 shows the layout of the LOCAL and NOCAL control record areas. Two sectors are allotted for each area. The word count is the number of words used to store (1) the 1-word word count, (2) the 2-word mainline name, and (3) the names of the LOCAL/NOCAL subprograms applying to the mainline name, each two words. This format is repeated for each mainline name encountered in the *LOCAL and *NOCAL records.

Phase D then re-calls Phase C if a type change was detected or Phase B if the last of the Supervisor control records was detected.

Phase D, once loaded, reads and prints the Supervisor control records until a type change or the last control record is detected.

CAL: Initializes the disk output area pointers and switches; contains a routine to read, print, and test the record type of the input record.

NAMEA: Edits and stores the mainline name in the disk output area; adjusts the pointers, switches, and counts for the disk output area and the record input area.

SUB: Edits and stores the subroutine name in the disk output area; adjusts the pointers, switches, and counts for the disk output area and the record input area. If the next character is not a comma or a blank, it is invalid. A blank indicates the end of the subroutine names applying to the current mainline name. If a comma is followed by a blank, the routine looks for a continuation record. Otherwise, it branches to SUB to process the next subroutine name.

NAMER: Stores each character of the mainline or subroutine name in a separate word for conversion to modified EBCDIC; compresses the name into name code.

RIGHT: Looks up a single character in the EBCDIC table (TAB1). When the input character is verified, it is left-justified and truncated into modified EBCDIC code. VERT1 contains the converted character.

DOFLO: Detects a disk buffer overflow. If the number of words used to store *NOCAL or *LOCAL records exceeds 640, an error is indicated by a printed message and the program returns to Phase A.

WNDUP: Initializes to write the disk output area to disk storage; initializes to read Phase C back into core storage and to re-enter Phase C at LTST.

ENDUP: Initializes to write the disk output area on disk storage; initializes to read Phase B back into core storage and to re-enter Phase B at SAVA.

Phase E

Chart: AH

- Edits and converts the file number to binary and stores the number.
- Converts the file name to name code and stores the name.
- Writes the file names and numbers on the disk.
- Reads and prints all FILES records (after the first).
- Calls Phase C at a type change or Phase B at the end of the Supervisor control records.

Phase E is initialized and called by Phase C to process *FILES records.

The file name is extracted and converted to name code and stored in the disk output area. The

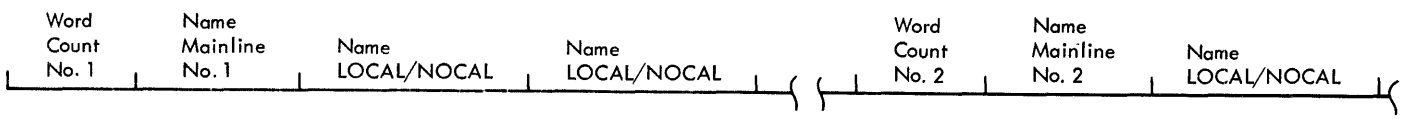


Figure 2. Layout of the LOCAL and NOCAL Control Record Areas

file number is also extracted, converted to binary, and then stored in the disk output area. Upon detection of a control record type change or the end of the Supervisor control records, Phase E writes the disk output area (the file names and numbers) in the FILES control record area of disk.

Figure 3 shows the layout of the FILES control record area. Two sectors are allotted for this area. The word count is the number of words used to store (1) the 1-word word count, (2) a 1-word file number for each file designated, and (3) a 2-word file name for each file designated.

Phase E then recalls Phase C if a type change was detected or Phase B if the last of the control records was detected.

Phase E, once loaded, reads and prints the Supervisor control records until a type change or the last control record is detected.

FILES: Initializes the disk output area pointers and switches; contains a routine to read, print, and test the record type of the input record.

NUMER: Edits each position of the file number. Each character is verified as numeric and the required left parenthesis is detected. The number is converted to binary and is stored in the disk output buffer.

NAME: Edits each position of the file name. Each character is verified as valid alphameric and the required right parenthesis is detected. The individual characters are converted to a modified EBCDIC and then are compressed into name code. The compressed name is stored in the disk output buffer.

NTBLN: Indicates an error if the position following the right parenthesis is other than a comma or blank. An appropriate message is printed and the program returns to Phase A. A comma followed by a blank indicates a continuation record and the program branches back into the FILES routine. A comma followed by a left parenthesis branches the program back within NUMER to process the next entry.

SLDR: Right-justifies the file number before it is converted to binary.

DOFLO: Detects a disk buffer overflow. If the number of words used to store the file names exceeds 640, an error is indicated by a printed message and the program returns to Phase A.

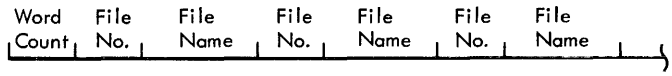


Figure 3. Layout of the FILES Control Record Area

WNDUP: Initializes to write the disk output area to disk storage; initializes to read Phase C back into core storage and to re-enter Phase C at LTST.

ENDUP: Initializes to write the disk output area on disk storage; initializes to read Phase B back into core storage and to re-enter Phase B at SAVA.

THE LOADER

There are two entry points to the Loader, one for Disk System format loads and the other for Core Image format loads. The format in which the object program is stored on the disk (Disk System format or Core Image format) determines which of these entry points will be entered from the Supervisor or DUP.

The appropriate entry point is selected either by Phase B of the Supervisor after detection of a XEQ Monitor control record or by the Disk Utility Program (DUP) after detection of a *STORECI DUP control record. Detection of either of these control records causes the controlling program to perform the following operations prior to the calling and execution of the loading program:

An XEQ control record causes the Supervisor to store in COMMA the mainline name, the code for the disk I/O version requested, and the indicator which causes the Loader to print a storage map.

A *STORECI control record causes DUP to store in COMMA the mainline name and the code for the requested version of disk I/O. In addition, the DUP program sets switches in COMMA which cause the Loader to print a storage map and to return to DUP after the core load is built.

The Supervisor writes *LOCAL, *NOCAL, and *FILES records in a special area on the disk. (See the Supervisor, Phase C, for the format of these records.) If DUP detects a *LOCAL record, an error message will be printed.

If the mainline name appears in the control record, it is located in LET/FLET (Location Equivalence

Table). The mainline name must appear in the *STORECI control record. The mainline name must appear in the XEQ control record if the program to be loaded is in Core Image format or is located in User Storage. Hence, if the mainline name does not appear in the XEQ control record, it is assumed that the program is located in Working Storage and is in Disk System format. From LET/FLET the disk block address is computed and stored in COMMA. Also, the format, i.e., Core Image or Disk System, of the mainline program is determined.

The CIB

Upon every entry to the CALL LINK entry in the Skeleton Supervisor, the contents of core storage between locations 256 and 4095 are saved on the CIB. This area is assumed by the Supervisor to be COMMON. This constitutes exactly twelve 320-word sectors, which are written on sectors three through fourteen of the CIB.

As a core load is built, the first sector of the CIB is used by the Loader to build up the Core Image Header record during a DSF load. In this same type of load the first word of the core load, if it is to reside below core location 4096, is placed in the first word of the second sector of the CIB, followed by consecutive words of the core load until the core load is complete, except for those words which should reside (at execution time) at core locations greater than 4095. The Loader, in building the core load, overlays as necessary the saved COMMON, sectors three through fourteen.

Thus, at the end of the loading process, the CIB contains the Core Image Header Record, that part of the core load which is to reside below core location 4096, and any part of COMMON which is to reside below core location 4096.

If the core load was built as a result of an XEQ control record, Phase 8 will first convert the number of words of COMMON in the CIB to a sector count, rounding the count up by one if there is not an integral number of sectors. This number of 320-word sectors will then be read directly into core storage from the disk, thus restoring COMMON. Then the core load itself is read directly into core storage from the CIB, except for the contents of the second sector of the CIB, which is read in by the Skeleton Supervisor.

To take an example, suppose that a link to a program with an object program of 626 words has occurred. The object program is to reside at core location 450, and the object-time transfer vector is 25 words long. A COMMON of 3004 words has been defined. Consequently, the core load consists of 641 words (626+25), which is two full sectors plus one word. COMMON occupies ten sectors (3004÷320=9+≐10). Phase 8 will first read the fifth to the fourteenth sectors of the CIB into core, beginning at core location 896 (COMMON actually begins at location 1092). Next, all 320 words of the third sector and one word of the fourth sector will be read into core, beginning at location FF0. At this point all COMMON has been restored (locations 1092-4095) and all but the first 320 words of the core load have been read into core (locations FF0-1091). The Skeleton Supervisor performs the last step of the process, which is to read the second sector of the CIB into core locations 450-F69 and to transfer control to the object program.

Figure 4 shows relative allocation of core storage during Loader execution.

DISK SYSTEM FORMAT LOADING

Loading (relocating) from Disk System format requires nine phases of the Loader (0 thru 8) plus the routines to print error messages and a storage map

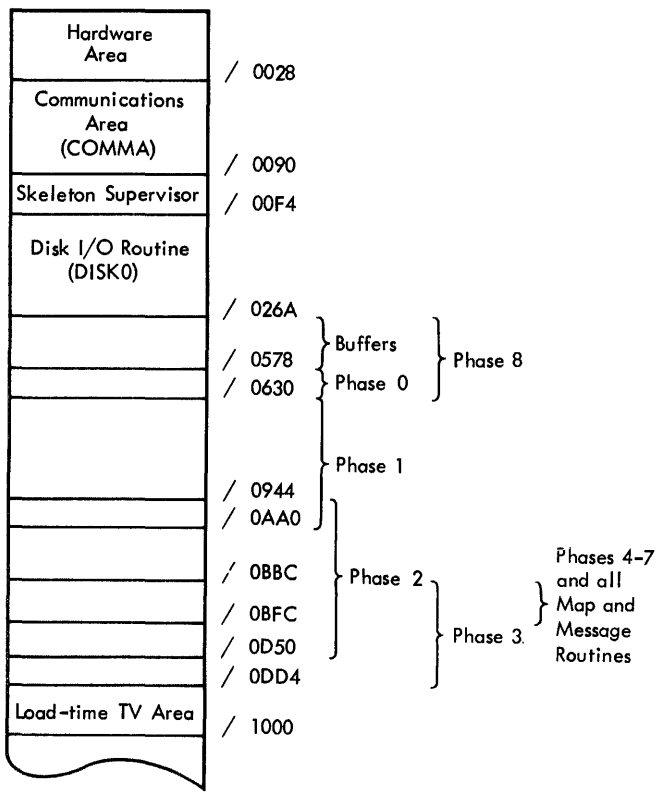


Figure 4. Storage Map of the Loader

(see Chart AJ). The mainline to be relocated can reside in either Working Storage or User Storage.

Phase 0 (When entered for Disk System format loads)

The entry point is BP100. This phase simply loads Phase 1 and transfers control to it.

Phase 1

- Initializes the processing of the *LOCAL and *NOCAL records.
- Builds the load-time TV and stores the first entry, the TV entry for the requested disk I/O routine.
- Processes the mainline header record.

LK Controls the reading of a given number of words from disk storage to core storage.

NW Fetches the next data word in sequence from the data buffer and, if required, reads the next sequential sector.

RH Reads into the data buffer the sector containing the header record for both mainline and subprograms.

GP Reads or writes one disk sector. The operation to be performed, i.e., GET or PUT, is determined by the entry point to the routine.

BT Builds the load-time transfer vector (TV). The first entry is always for the disk I/O routine requested. Following this, as LIBF and CALL statements are encountered by the Loader, additional entries are made to the TV.

The first entry in the TV occupies words 4086-4089. Subsequent entries are stored in successively descending blocks of four words each (see Load-Time TV).

LS Searches LET/FLET for program and data file names. If the name is not found, the load is terminated. If the name is found, the output from this subroutine is the disk block address of the program or data file. For core image programs the execution address, the loading address, and the word count are also a part of the output.

PM Prints the storage map if requested. Error messages are printed as errors are encountered during the loading functions.

TL Exits to the Monitor call routine or to the DUP program, depending upon the entry point used.

EX Exits through routines PM and TL. This routine is entered for those errors which cause the Loader to terminate the loading process.

MC Extracts from the mainline header record the addresses, counts, and indicators required for loading a mainline program. This routine utilizes the BT routine to make the initial entry in the load-time TV, a LIBF entry to the disk I/O routine requested by the user (DISKZ, DISK1, etc.). If no specific version is requested, DISKZ is used. After all *LOCAL and *NOCAL records have been processed, the routine reads in and transfers control to Phase 2.

LN Examines all *LOCAL and *NOCAL records in the LOCAL/NOCAL sectors and enters the subprogram names in the load-time TV. All LOCAL subprogram types are checked to determine if they are valid for LOCALs, i.e., the LOCALs are not mainlines or interrupt level (IL) subroutines.

Phase 2

- Relocates and converts the mainline and all subroutines and subprograms from Disk System format to Core Image format.

MC Controls the operation of Phases 3 through 8. After the execution of Phase 6, this routine returns to DUP if the Loader was called by DUP as the result of a *STORECI control record. Otherwise, this routine calls Phase 7, executes it, and then calls the Phase 8 and transfers control to it.

RL Controls the conversion of all programs comprising a core load from Disk System format to Core Image format. As a program is converted, the absolute address of each entry point is placed in the third word of its load-time TV entry. LIBFs within the program are replaced by a short BSI instruction with a tag of 3 and a displacement to the corresponding LIBF TV entry. CALLs within the program are replaced by a long indirect BSI instruction, the second word of which is the execution-time address of the corresponding CALL TV entry.

The RL routine replaces DSA (define sector address) statements with a sector address, word count, and entry point (this will be zero for data files) from LET/FLET. In addition, the absolute addresses of device servicing routines are inserted into all IL subroutines required by the particular core load.

TR Places the core load being built, one word at a time, into the Core Image Buffer (CIB) or into core storage. If the address at which a word is to be stored is greater than 4095, the word is placed directly into storage. If this address is greater than the capacity of the machine, an error message is printed. If the address at which a word is to be stored is 4095 or less, the word is stored in the Core Image Buffer. Thus the core load can be entirely in core storage, entirely in the Core Image Buffer, or divided between the two.

WR Writes the core load being built, one sector at a time, on the CIB. A disk write occurs when the address at which a word is to be stored falls outside the limits of the one-sector buffer which is contained in core. This subroutine also writes LOCALs and SOCALs in Working Storage.

XC Places the core address of the LIBF TV entry associated with a subprogram entry point into the exit control cell for that entry point. For all entry points referenced by LIBF statements the address of the exit control cell is the address of the subprogram entry point +2. For example: if the entry point FLOAT is located at the address 1000_{10} and the corresponding LIBF TV entry is located at the address 4075_{10}, then the XC routine places the address 4075_{10} into location 1002_{10}. This operation provides for execution time return linkage through the link word contained in the LIBF TV.

MV Moves the DEFINE FILE table to a processing area (see DF) and, when processing is complete, saves the table in the Core Image Buffer.

DF Places into the table entry for a given Defined File the sector address assigned to that file. This address can be an absolute sector address taken from LET/FLET or a sector address relative to the beginning of Working Storage. In the latter case the address is calculated and assigned by the DF routine.

If the DEFINE FILE table specifies a disk block count for a file defined in User Storage that is greater than the disk block count for that file contained in LET/FLET, the count from LET/FLET replaces the count in the DEFINE FILE table.

If only one file is defined in Working Storage and if the disk block count for that file exceeds the available Working Storage, the count in the DEFINE FILE table is reduced to the length of Working Storage. If multiple files are defined in Working Storage and if the total disk block count exceeds the available Working Storage, the core load will not be executed.

CK Checks to ensure that COMMON does not extend into the area to be used by Phase 8. An overlap results in an error message. Loading continues, but the non-XEQ and non-DUP switches are set.

ML Checks to ensure that the loading address for the mainline is greater than the highest core location occupied by the requested version of Disk I/O.

Phase 3

- Controls the loading of subprograms by class.
- Processes the program header record of all routines named in the load-time TV.
- Selects and controls the loading of required IL subroutines.

IL Selects and relocates the IL subroutines associated with each of the required interrupt levels within a particular core load.

HR Extracts the data required for loading (e.g., precision, type, and entry point names) from the header records of both mainline programs and subprograms.

CC Controls the loading of subprograms by class. The in-core routines are loaded first, followed by LOCALs and then SOCALs. The latter routines will be loaded according to system overlay level if system overlays are used. See System Overlay Scheme.

TY Checks that subprograms requiring an LIBF reference are referenced by LIBFs and that subprograms requiring a CALL reference are referenced by CALLs.

SV Scans the load-time TV twice, first to ensure that the routine has not been previously

loaded and then to find any other entry points to the routine being relocated.

The first scan examines the entry points in the load-time TV which precede the current one. If another entry point to the same routine is found, the routine has been loaded and the absolute address of the current entry point in that routine is placed in the third word of its load-time TV entry. The routine is not loaded a second time. If no other entry points are found in this scan, the routine is loaded.

The second scan examines the entry points following the current one. If other entry points to the routine being relocated are found, the absolute addresses of each of those entry points are stored into the third word of their respective load-time TV entries.

Phase 4

- Checks to see if the core load built in Phase 3 fits into core storage.
- If LOCALs are used, computes the size of the Flipper table and decides which of the two Flipper routines is required, i.e., FLIP0 or FLIP1.
- Initiates the attempt to fit oversize core loads into core storage through the use of overlays.

ET Calculates the amount of storage required for the core load. If the core load fits into the available storage, control is returned to Phase 2 (MC) which reads in Phase 6 to construct the object-time TV.

If the core load does not fit, further processing is required.

See System Overlay Scheme.

Phase 5

- Outputs the Flipper table and Flipper routine if LOCALs are present.
- If SOCALs are required, outputs a special TV for any 2-word calls (functionals) which are a part of SOCALs.
- Establishes the class code for loading SOCAL subroutines.

LT (Executed if LOCALs are present) outputs the Flipper table (parameters required for loading and execution of LOCALs) and then outputs the selected Flipper routine. LT sets an indicator which causes Phase 2 to to load the LOCAL subprograms. Phase 2 scans the load-time TV and adds to the core load all LOCAL subprograms referenced.

ST (Executed if no LOCALs are present or if they have already been processed) returns to Phase 2 if no SOCALs are required. If SOCALs are required, this routine sets an indicator to cause the loading of the next SOCAL by class code.

See System Overlay Scheme.

Phase 6

- Builds the object-time LIBF and CALL TVs.
- Ensures the odd boundary for the Floating Accumulator (FAC).
- Completes the Core Image Header record.

ER Builds one object-time TV for LIBFs and one for CALLs. (See Object-time TV.) The Flipper table address is placed in the TV entry of all LOCALs.

If it is necessary, a dummy entry is made in the CALL TV by this routine in order to make the address of the rightmost word of the Floating Accumulator (FAC) an odd address.

The Core Image Header record is completed by this routine and is then stored in the first sector of the Core Image Buffer.

Phase 7

- Loads the requested disk I/O routine into core.
- Saves part of the Skeleton Supervisor and COMMA on the disk if DISKZ has been requested.
- Moves the interrupt TV for the core load to be executed into the Hardware Area in low core.

LD Tests an indicator in COMMA which indicates the user-requested version of disk I/O

and then reads that disk routine into core. If DISKZ is requested, part of the Skeleton Supervisor and COMMA is written on the disk in order to allot more storage to the mainline program. That portion of the Skeleton Supervisor and COMMA which was overlayed is restored before control is returned to the Monitor after execution. No disk I/O loading occurs if DISK0 is called, because this routine is used by the loader and therefore is already in the disk I/O area.

NOTE: Phase 7 has its own disk I/O subroutine for reading the user's disk I/O routine into the disk I/O area.

Phase 8

The description of this phase is identical to that which is given in the section on Core Image Format loads except that, for DSF loads, Phase 8 reads into core storage only that part of the core load which is in the CIB, including COMMON, before transferring control to the Skeleton Supervisor.

NOTE: Phases 7 and 8 are used in Disk System Format Loads only when the relocated program is to be executed. See routine MC in Phase 2.

CORE IMAGE FORMAT LOADING

For loading programs in Core Image format only three phases of the Loader are required: Phase 0, Phase 7, and Phase 8 (see Chart AK). Phase 0 processes the Core Image Header record and controls the fetching and transfer to Phases 7 and 8. Phase 7 returns control to Phase 0, whereas Phase 8 returns to the Skeleton Supervisor.

Phase 0 (When entered for Core Image format loads)

The entry point is BP200.

LK	Controls the reading of a given number of words from disk storage to core storage.
GET	Performs the disk read function.
BP	Extracts the parameters from the Core Image Header record and transfers them to COMMA. It fetches Phase 7 and transfers control to it. It then fetches Phase 8 and relinquishes control to it.

Phase 7 (Same description as in the section Disk System Format Loading.)

Phase 8

First restores COMMON from the CIB, if any, if the program being loaded is a CALL LINK. Its other function is to read all but the first sector of the core load into core. It sets up the sector address and word count of the first sector and relinquishes control to the Skeleton Supervisor, which it has supplied with the necessary information for moving the object-time TV into its execution-time location. The Skeleton Supervisor then completes the loading process and transfers control to the object program.

Figure 5 shows the relative allocation of core storage at user execution time.

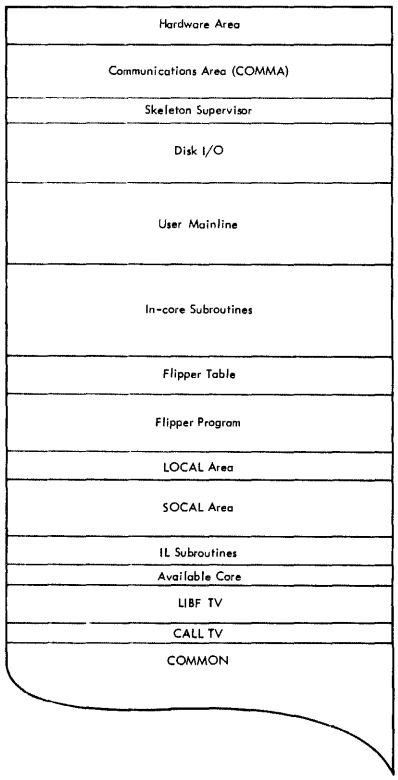


Figure 5. Storage Layout at Object-Time

SUBROUTINES USED BY THE LOADER

Most phases of the Loader use the following routines. These routines are located on the disk, and when called are loaded into the LET search buffer. This buffer occupies the first 320 words of the Loader.

1132/Console Printer Print Routine - Performs any printing by the Loader -storage map, error messages, etc.- on the principal print device. The 1132 Print Routine is loaded at system load time only with systems having an 1132 Printer; otherwise the Console Printer Print Routine is loaded.

Error Message Routines - Set up the appropriate error messages for printing. See the publication IBM 1130 Monitor System Reference Manual (Form C26-3750) for a listing of these messages and conditions which cause them to be printed.

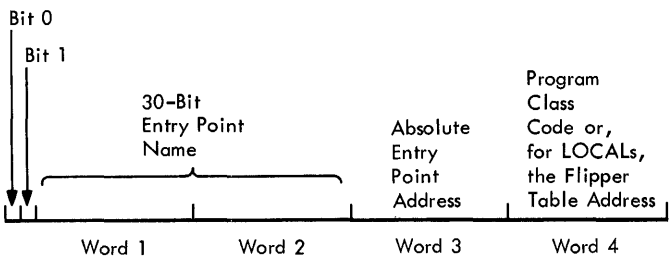
Map Routines - Set up the titles, messages, etc. required for the printing of the storage map.

THE LOAD-TIME TRANSFER VECTOR

The load-time TV consists of an entry for:

1. The Disk I/O routine specified by the user on the XEQ or STORECI record.
2. Each LOCAL and NOCAL entry point specified on a *LOCAL or *NOCAL record.
3. Each different CALL or LIBF reference in the relocated core load.
4. If System overlays are employed, one special entry for each overlay.

Each entry in the load-time TV is four words in length. The first entry is stored in locations 4086-4089, the second in 4082-4085, etc.



During the first load-pass, the first two words of each TV entry contain the symbolic name of the entry point associated with the entry. This 30-bit name is right-justified in the 32 bit positions of the two words. LOCALs are flagged in the TV by setting bit zero in the 32-bit name. Bit one is set to indicate NOCALs and those entry points referenced by CALL statements.

NOTE: All subprograms indicated as NOCALs must be type 4 or 6 routines.

Phase 1 makes the TV entries for the Disk I/O routine (in routine MC) and for LOCAL and NOCAL subprograms (in LN). The entry for the Disk I/O version is made, even if the program contains no LIBF statements to a Disk I/O routine. Phase 1 also sets bits zero and one of each entry (as necessary) as the entry point name is added to the TV.

The third word of each load-time TV entry, initially zero, ultimately contains the absolute core address at which the corresponding entry point will found at execution time.

A non-zero value in this word indicates to the Loader that the routine associated with this TV entry has already become a part of the core load and thus is not to be loaded a second time. This absolute address is added by either Phase 2 (routine RL) or Phase 3 (routine SV).

The fourth word, also initially zero, is reserved for the class code of the routine to which the load-time TV entry corresponds. This code is used in determining the order in which subprograms will be loaded if SOCALs are employed. This code is inserted by Phase 3 (routine SV).

The class 0 subprograms are subtype 0 subroutines of types 3, 4, 5, and 6. See the publication IBM 1130 Monitor System Reference Manual (Form C26-3750) for descriptions of the type and subtype specification.

These subprograms are termed "in-cores" because they are loaded with all mainlines. The IL subroutines are also "in-cores" but they are never a part of any overlay and they technically do not belong to a class.

The class 1 subprograms (System Overlay 1) are the Arithmetic and Functional subprograms, which comprise the first SOCAL.

The class 2 subprograms (System Overlay 2) are the FORTRAN I/O, and I/O conversion routines which comprise the second SOCAL.

Disk FORTRAN I/O is the only class 3 subprogram. It, along with a 320-word buffer, comprises the third SOCAL.

On the first load-pass the mainline is loaded, followed by the subprograms in the order of their appearance in the TV. If the core load fits and no LOCALs are specified, the core load is established as it is. Phase 2 then calls Phase 6 to build the object-time TV.

If the core load does fit and if LOCALs are specified, a second load-pass is made. In this case, all subprograms except the LOCALs are considered as class 0 subprograms. Thus, the mainline is loaded, followed by the class 0 (all) subprograms, followed by the Flipper table and Flipper routine. The LOCALs are written out on Working Storage following any Defined Files.

If the core load does not fit into the available storage as determined by Phase 4, a second load-pass is made. In this case, during the second load-pass, the mainline is loaded, followed by the class 0 subprograms. If LOCALs are present, the Flipper table and Flipper program are loaded next, followed by the LOCAL subprograms, which are written out on Working Storage. After this, the remaining subprograms are loaded, i.e., written out on Working Storage following the LOCALs, by class code. See System Overlay Scheme.

For a LOCAL subprogram the fourth word of the TV entry contains the address of the Flipper table entry for that LOCAL. Phase 5 places word three of the TV entry into the corresponding Flipper table entry. Phase 6 then moves word four of the TV entry into word three. Thus, at execution time, the TV entry causes control to pass to the Flipper routine through the Flipper table rather than to the called subprogram (see Flipper Table).

THE FLIPPER TABLE

The Flipper table and Flipper routine become part of a core load only if LOCAL subprograms are specified by the user for that core load.

The Flipper table consists of a 6-word entry for each of the entry points specified in an *LOCAL record which is referenced by a CALL statement and a 5-word entry for each entry point referenced by an LIBF statement.

The word count, sector address, and absolute entry point are computed and inserted into each Flipper table entry by the Loader (Phase 5) as it processes each LOCAL. Phase 5 also builds the linkage to the Flipper routine (a long BSI instruction) and, for CALL entry points, a linkword.

The LOCAL subprograms are placed into the Working Storage area on the disk following the Defined Files, if there are any.

The Flipper routine is the subroutine which, at object-time, using the parameters of the Flipper table entry, reads a LOCAL subprogram when it is called from Working Storage into the LOCAL overlay area, and transfers control to it.

A special Flipper table is created for SOCALs if the System Overlay scheme is employed. See System Overlay Scheme.

THE OBJECT-TIME TV

Phase 6 of the Loader builds two separate object-time TVs: the CALL TV and the LIBF TV.

Each CALL TV entry is a single word containing the absolute address of a subprogram entry point. However, in the case of a LOCAL subprogram referenced by a CALL statement, the absolute address is the address of the corresponding Flipper table entry instead of the subprogram entry point.

Each LIBF TV entry is comprised of three words. Word one is the linkword. Words two and three contain a long BSC instruction to the subprogram entry point. However, in the case of a LOCAL subprogram referenced by an LIBF statement, words two and three contain a long BSC instruction to the corresponding Flipper table entry instead of the subprogram entry point.

The LIBF TV is preceded by two special entries, each three words in length. The first is the Floating Accumulator (FAC). The address of the first word of FAC must be an odd address. Therefore, if necessary, a dummy entry is made in the CALL TV by Phase 6 in order to make FAC begin at an odd address.

The second special entry is one 3-word entry for use by certain subroutines to indicate overflow, underflow, and divide check.

If the System Overlay scheme is employed, the object-time LIBF TV contains special entries for SOCAL subprograms referenced by LIBF statements. These entries transfer indirectly either to the referenced subprogram if the overlay containing the subprogram is presently loaded or to the SOCAL Flipper in order to load the required overlay and transfer to the referenced subprogram. See System Overlay Scheme.

The object-time CALL TV does not contain entries for SOCAL subprograms referenced by CALL statements, i.e., functionals, if a System Overlay is employed. See System Overlay Scheme.

Figure 6 shows the layout of the object-time TVs.

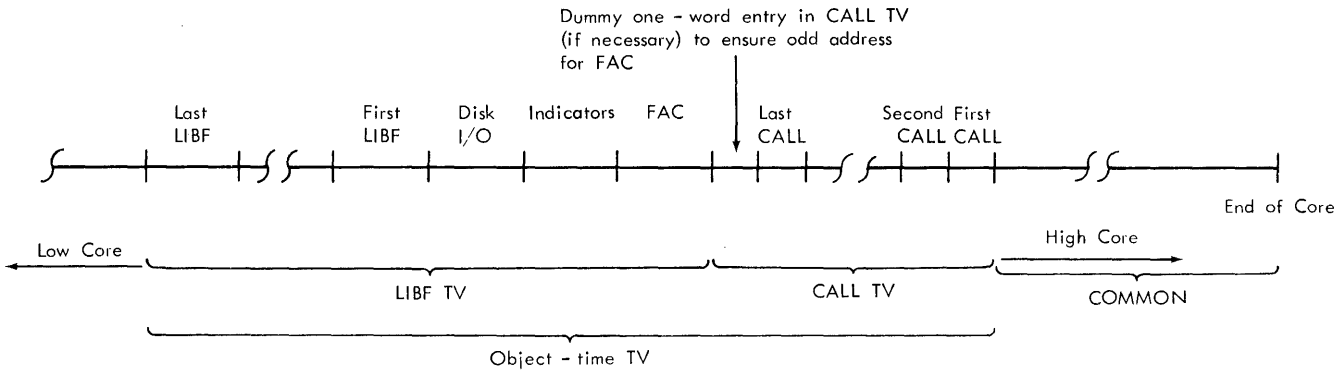


Figure 6. Layout of the Object-Time TV Area

SYSTEM OVERLAY SCHEME

If, after the first load-pass, Phase 4 determines that the core load will not fit into the available storage, and if the mainline is an Assembly-written program, loading is terminated and a message is printed indicating by how much the core storage capacity has been exceeded.

If the mainline is a FORTRAN program, Phase 4 initiates a second load-pass. Phase 2 reloads the mainline program and the class 0 or "in-core" subprograms. If LOCAL subprograms are present, Phase 5 (routine LT) indicates their presence to Phase 2 which then loads the Flipper table, the Flipper routine, and the LOCALs. Phase 4 then attempts to make the core load fit by overlaying the class 1 SOCALs (the arithmetics and functionals), the class 2 SOCALs (FORTRAN I/O, I/O, and I/O conversion routines) and the class 3 SOCALs (Disk FORTRAN I/O plus a 320-word buffer). (This third overlay is made only if Disk FORTRAN I/O is called.)

If Phase 4 finds that the core load can be made to fit, it actually causes the Loader to create the overlays described above. Otherwise loading is terminated and an error message printed out.

A special Flipper table is created for SOCALs. This table, in conjunction with the DISKZ routine, performs the same function for SOCAL subprogram references as do the standard Flipper table and Flipper routine for LOCAL subprograms. However, this Flipper table is contained within the DISKZ routine in the disk I/O area.

The CALL TV does not contain entries for SOCAL subprograms referenced by CALL statements, i.e., functionals. The one-word CALL TV entries are attached to the front of the System overlay in which the corresponding subprograms appear. Since the subprograms in overlays 2 (FORTRAN I/O, I/O, and I/O conversion routines) and 3 (Disk FORTRAN I/O routines) can be referenced only by LIBF statements, the one-word entries preceding these overlays all cause a return to the SOCAL Flipper to load the arithmetic/functional overlay.

Suppose a core load (1) contained a FORTRAN-written main-line, (2) required all three System Overlays to fit into core, and (3) contained references to FSQR and SIN. Then each overlay would start with a special CALL TV two words in length, one word for FSQR and the other for SIN. Any user-written routines referenced by CALL statements would be represented in the normal CALL TV which is found just to the left of COMMON. The two words of the CALL TV for System Overlay 1 would contain the actual core addresses of the entry points FSQR and SIN. The two words of the CALL TV for System Overlays 2 and 3 would contain the addresses of the SOCAL flipper entries for FSQR and SIN. All CALL FSQR/SIN statements would have been replaced with long, indirect BSI instructions to the address of the corresponding special CALL TV entry. Thus, at execution time whenever a CALL FSQR/SIN is encountered, a branch will be made to either FSQR/SIN or to the SOCAL flipper. In the latter case the flipper would first read System Overlay 1 into the overlay area and then re-execute the branch to FSQR/SIN. In either case FSQR/SIN would be entered and executed.

Figure 7 shows the CALL TV for the above example.

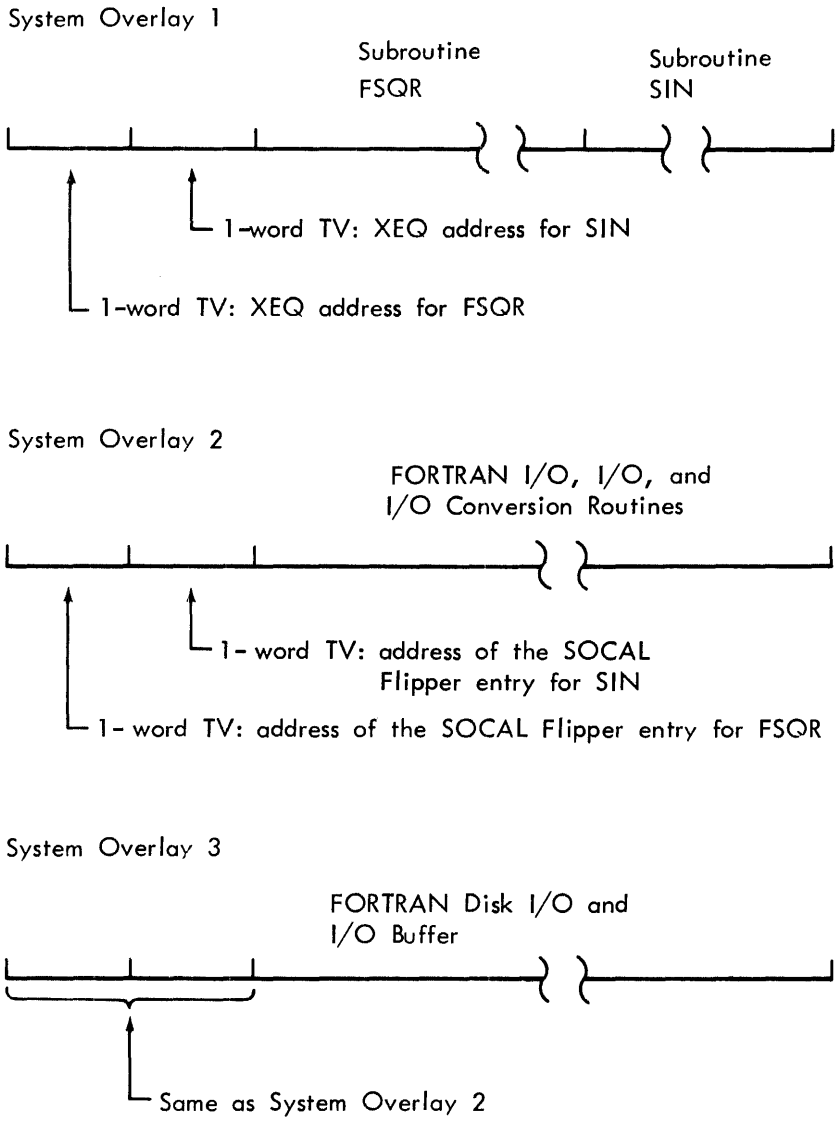


Figure 7. Format of the CALL TV for System Overlays

INTRODUCTION

The Disk Utility Program is designed to accomplish the following:

- Automatically allocate disk storage to each program assembled or compiled.
- Make these programs available in card or paper tape format.
- Print out programs and certain other predetermined areas from disk storage.
- Provide automatically printed records of the status of the size of the User Area and Work Storage Area.
- Provide automatic file protection for all areas other than Working Storage.
- Provide the facility to delete the Assembler Program and/or FORTRAN, and to specify and enlarge the Fixed Area of Disk Storage.
- Provide various other disk and core maintenance operations.

The Disk Utility Program (DUP) is called into operation when the Supervisor recognizes a // DUP record. One sector of DUP, DUP Common (DUPCO), is brought into core storage. DUPCO calls in DUP Control (DCTL). DCTL calls in the principal print device routine (VIPX or TYPX) to print as required. Following this, DCTL calls in the principal input device routine (CARDX or PTX) to read the next record, which should be a DUP control record.

The DUP control record is then printed, decoded, and checked for accuracy. Switches are set in DUPCO in accordance with information obtained from the control record. The required DUP function is then called from disk, overlaying the core area of DCTL as required.

When DCTL transfers control to other DUP functions, LETAR, the buffer used in the LET/FLET search, contains the sector of LET/FLET last read from disk storage; i.e., the portion of LET/FLET containing the entry involved.

Control is turned over to the DUP function which performs its assigned tasks according to the information that was extracted from the DUP control record.

Upon completion, the function returns to DUPCO, which calls DCTL back in. DCTL calls in the principal print routine and prints the DUP EXIT message. The principal input/output device routine is called in to read the next record.

This sequence of events is repeated until the principal input device reads a Monitor control record, in which case control is returned to the Supervisor. The Supervisor now examines the Monitor control record and acts accordingly.

Figure 8 shows the layout of storage during the execution of DUP. Chart BA is an overall flowchart of the DUP functions.

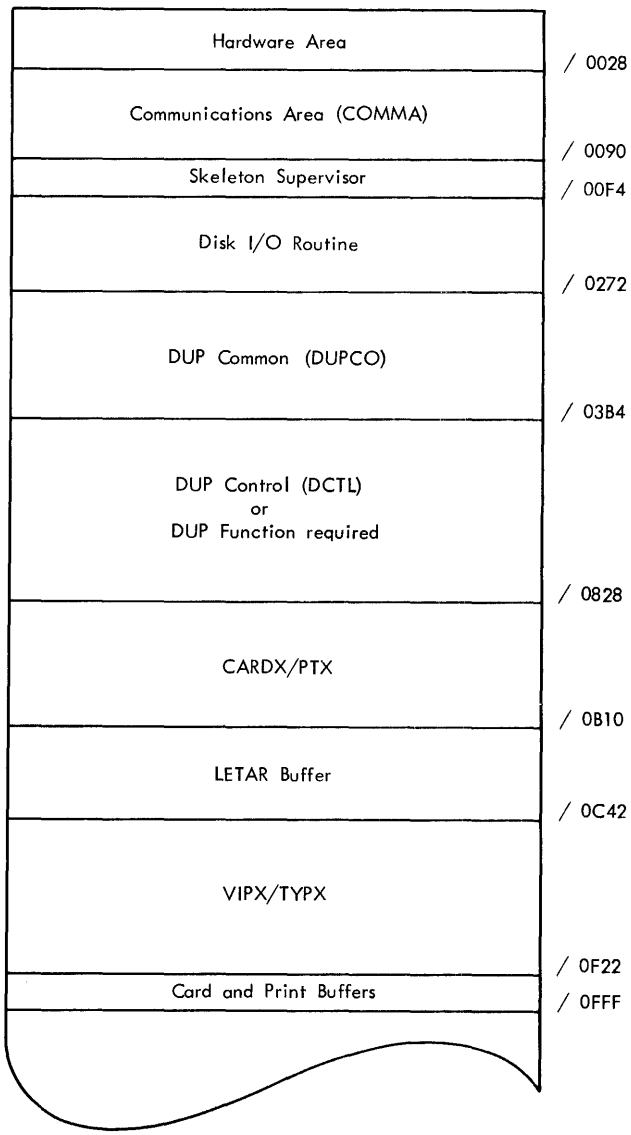


Figure 8. Storage Layout of DUP

DUP FUNCTIONS AND ROUTINES

DUP Common (DUPCO) Routine

Chart: BB

- Provides entry for Supervisor to DUP.
- Provides common conclusion of all DUP functions.
- Provides for the return of the Loader to DUP after performing a Disk System Format Load.
- Provides Disk multi-sector read/write capabilities for DUP.
- Provides common system check 'WAIT' for DUP.
- Provides a common call and linkage to a DUP error message routine (TERRC/PERRC).
- Provides and initializes switches available for all of DUP.
- Initializes the Interrupt Transfer Vector for DUP.
- Calls in the Store Core Image function of DUP.
- Provides Interrupt Level routines for DUP I/O routines.
- Calls in DUP Control (DCTL).

DUPCO is a one sector routine that is resident in core all the time that DUP is in control, except while the Loader is converting a Disk System format program into Core Image format for DUP.

Switches and routines that are common to many of the DUP functions are kept available in core at all times. Because of this, DUPCO provides many entry points from other DUP functions as well as from the Supervisor and the Loader.

An initialized DUPCO is read in by the Supervisor when a DUP control record is read and the non-DUP Switch (word 64_{16}) is zero.

A non-initialized DUPCO (DUPCO that is written to the disk before calling the Loader) is read back in by the Loader to continue the Store Core Image function.

If DUPCO is entered with a BSI to REST, REST is non-zero and an exit message will be printed. If the entry is a BSC to REST +1, the message will be inhibited.

All DUP disk read/write operations are done through the multi-sector routine in DUPCO. This provides a maximum word count of 320 words to the DISK0 routine and keeps supplying word counts until the original word count requested by the DUP routine has been transmitted.

NOTE: IOAR Header, used throughout this description of DUP, refers to the two words required to be in front of all core areas at the time they are used as buffers for Disk I/O operations. These two words are the word count and the sector address respectively.

Figure 9 shows the storage layout of the DUPCO routine.

Entry Points

NYETC	Selects blank record. Exit to RDCTL. Calling address in NYETC.
SUPDC	Supervisor entry to DUPCO. Supervisor has read DUPCO to core from disk. Exit to SEPT.
REST	DUP functions return to this point in DUPCO after completing required operations. Exit to REST2. Return address in REST if valid DUP function completed.

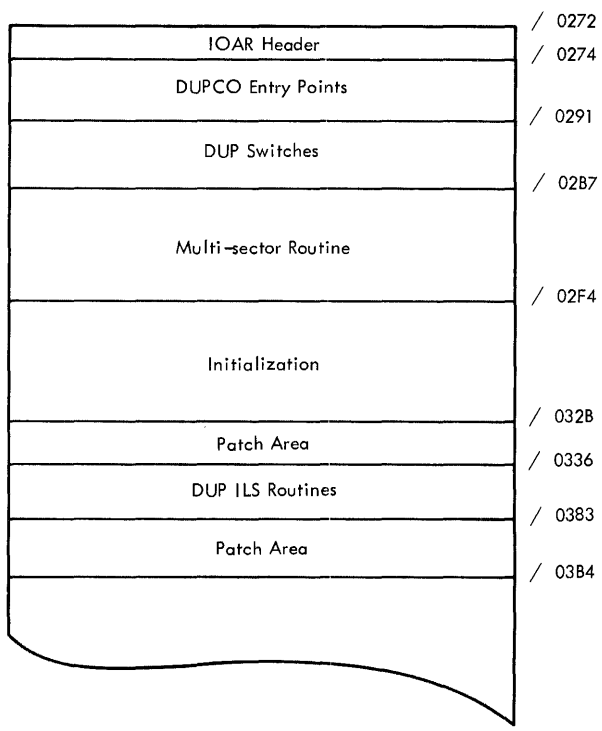


Figure 9. Storage Layout of DUPCO

REST+1	Entry point generally from error point routines whenever a DUP function is not completed successfully. Printing of the exit message is inhibited when REST is left zero. Exit to REST2.
LDRDC	Loader entry after converting the program, requested by DUP, to Core Image format and leaving the Core Image Program in the Core Image Buffer (CIB) and/or core storage. Exit to RLEPT.
PUT	DUP system entry to multi-sector routine for writing information to disk. All DUP functions except DWADR use this entry for writing to disk. Requires even address for IOAR Header. XR3 must contain address of IOAR Header. Return Address in PUT. Exit to PUT2.
GET	DUP system entry to multi-sector routine for reading information from disk. All DUP functions use this entry for reading from disk. Requires even address for IOAR Header. XR3 must contain address of IOAR Header. Sets up disk call for read operation. Moves return address to PUT. Exit to GOMUL.
SYSCK	DUP system entry if an error has occurred that may require such drastic action as reloading the pack. Used primarily for debugging. Will stop absolutely regardless of interrupts. Interrupts will, however, be serviced in the normal manner. Return address in SYSCK.
SEXIT	DUP system error entry to call error message routine. Move print error routine (PERR/TERR) from disk to core. Relative entry point in PERR or TERR for requested message in SEXIT. Exit to PERRC + 2 or TERRC + 2.

Multi-sector Routine

PUT2	Lifts file protection and sets disk call for a write operation. Exit to GOMUL.
GOMUL	Stores original word count and sector address, also checks for buffer address being even. Exit to SYSCK if not even, otherwise exit to SAVER.
SAVER	Saves the two words where the IOAR Header will be placed. Inserts the address of the IOAR Header into the disk call. A check is made to see if Working Storage is exceeded. If it has been exceeded, an error message is printed. Exit to MMORE if the word count is still more than 320. If the count is less than 320, load IOAR Header information to the accumulator. Exit to OUT2.
OUT2	Completes the storing of the IOAR Header and does required disk call operation and loops until ICNT has gone to zero. Exit to RSTRE.
RSTRE	Restores the two words previously saved to the IOAR Header area. Decrements the word count requested by 320. Exit to REFPA if zero word count. If the word count is still positive, the sector address is incremented by one. The location of the next IOAR header is adjusted by 320. Exit to SAVER.
MMORE	Forces a count of 320 into the IOAR Header setup (the requested count is still greater than 320). Exit to OUT2.
REFPA	Restores original file protection and restores original setting of XR3. Exit to the address in PUT.

Initializing Routine

REST2	Clears switches and indicators that are in DUPCO with the exception of ZEST, L,and those that precede L. Exit to SEPT.
RELPT	Initializes the Interrupt Transfer Vector (ITV) for DUP using closed subroutine FIRST. Calls in the Store Core Image function of DUP. Exit to the address specified in CISW (in Store Core Image Function).
SEPT	Uses routine FIRST to initialize the ITV. Exits to RDCTL.
RDCTL	Inserts the proper IOAR Header and calls in DUP Control. Exit to DUCTL in DUP Control.
FIRST	Closed subroutine to initialize Interrupt Transfer Vector (ITV). The address of the calling program is in FIRST. Exit to calling routine.

Interrupt Level Subroutines (ILS)

These routines contain the addresses that are put in the Interrupt Transfer Vector (ITV). The routines exit to the respective I/O routines, and, upon returning, turn off the interrupt level and return control to the instruction that was interrupted.

DUP Control (DCTL) Routine

Chart: BC

- Calls in the principal print device routine to print.

- Prints the DUP control record (ENTRY message).
- Prints the location and size of the program or area on which a DUP function was performed (EXIT message).
- Controls paper spacing after EXIT message.
- Calls in principal input device routine to read and punch.
- Reads DUP control records.
- Decodes DUP control records.
- Calls and searches LET and FLET for the name on the DUP control record.
- Calls in required DUP Function to complete action requested.
- Detects errors on DUP control records and prints the appropriate error messages.
- Returns control to the Supervisor when Monitor control record is recognized.
- Selects excess blank cards following a DUMP.
- Passes excess data cards following a STORE or STOREDATA.

DCTL is called in by DUPCO to determine what DUP is being asked to do. The principal print device routine is read from disk. The principal I/O device routine is also read from disk.

The DUP control record is then read from the principal input device and decoded. The function and the "FROM" and "TO" fields are converted from IBM card code to DUPCO switch settings. The NAME field is encoded from IBM card code to name code. The COUNT field is encoded from decimal to hexadecimal.

If the named function is DUMP, STORE or DELETE, a LET/FLET search for the name is done. An exit to SYSCK may occur if DCOM and COMMA or LET/FLET and COMMA don't agree. This situation can occur if manual intervention or a hardware failure interrupts some DUP operations while the table areas are being modified by DUP.

If STORE Core Image is requested, the subroutine FILEQ is called from disk to read, check, and record the following *FILES records, if any. The STORE function is then called directly from the FILEQ subroutine.

If no detectable errors occur, the required DUP function is read, overlaying DCTL, and control is turned over to the function.

If detectable errors occur, error messages are called from disk and the proper message is selected and printed using VIPX or TYPX. Then control passes back to DUPCO thru REST+1.

Entry Point

DUCTL	Enter from DUPCO. Call principal print device routine from disk into core storage. Exit to RDIO if BYESW is zero. Exit to PGMA.

I/O Operations

PGMA	Go to PRTBI to print EXIT message and return. Clear BYESW to inhibit EXIT message unless DUP operation completed. Control paper spacing depending on status of RPSW.
RDIO	Bring principal I/O device routine into core. Exit to FAKE.
FAKE	Requests the principal I/O device routine to read a record and convert its contents to packed EBCDIC code. The result is checked for blanks from right to left. If the entire record is blank, the exit is made to NYET9. The count of 80 minus non-blanks defines the number of characters that is to be printed in the ENTRY message. If columns 1 and 2 contain *D or *S, then exit will be to ELIM; otherwise exit to NYET.
ELIM	Goes to PRTEB to print the packed EBCDIC control record. Go to PRTCR before and after the printing of the control record to provide spacing the carriage before and after printing. Exit to DCODE.
NYET	If columns 1 and 2 are not //, exit to NYET2. Set Monitor non-read switch (word 66_{16}) to a negative number. Exit to SUPER; i.e., to the Skeleton Supervisor.
NYET2	Check columns 1 through 6 for *EDIT blank. If not present exit to FAKE, otherwise exit to CALLE.
NYET3,4,5,6 NYET 7, F, and T	Select appropriate error message address. Exit to NYET8.

NYET8 Insert error message address into print EBCDIC macro. Go to GET to bring in ERM to core from disk and return. (ERM contains Error Messages coded in EBCDIC for DCTL function only.) Print EBCDIC message by branching to PRTEB, then return and re-initialize the switches in DUPCO except for L, and those preceding L. Exit to Rest + 1 of DUPCO.

NYET9 Select stacker. Address of calling routine in NYET9. Exit to FAKE to read next control record.

Decode DUP Control Record Function Field

DCODE If columns 1 and 2 of the control record contain *D, exit to DDCTL. If they contain *S, then exit to STCTL. Otherwise exit to NYET.

DDCTL Scan columns 3 and 4 of the control record and exit to:
DFCTL if EF;
DLCTL if EL;
WACTL if WA;
NYET3 if not UM (invalid function field).
Examine columns 5 and 6 and exit to:
DPCTL if P blank;
DACTL if PD;
LECTL if PL;
LET2 if PF.
In none of the above exit to NYET3 (invalid function field).

Decode STORE Function

STCTL Scan columns 3 through 6 of DUP control record. Exit to NYET3 if not TORE. Set STSW in DUPCO. Decode column 11 for SOCAL information. Store numeric value of column 11 in T3MSW. Scan columns 7 and 8 for store modifiers. Check columns 7 and 8 and exit to:
SC13 if blank;
SCI if CI;
SDATA if DA;
SMOD if MO.
Otherwise exit to NYET3.

SCI Set Core Image switch in DUPCO. Force Loader to print core map. Set DTYPE in COMMA in accordance with type of Disk I/O specified in column 9. Exit to NYET3 if not legitimate character. Go to DACNT to record the number of *FILES records immediately following the *STORECI DUP control record, and return. Exit to SC13.

SDATA Exit to NYET3 if columns 9 and 10 not TA. Go to DACNT to put count in DATSW, and return. Exit to NYET7 if count field is blank. Otherwise exit to SC13.

SMOD Exit to NYET3 if not D blank in columns 9 and 10. Set SMODSW. Go to CKTMP to inhibit if in temporary mode, and return.

SC13 Check FROM field for card/paper tape or Working Storage. Check columns 13 and 14 and exit to:
FRIN if CD and Card is principal I/O.
FRPT if CD and Paper Tape is principal I/O.
FRPT if PT.
NYET4 if not Working Storage.
Set Working Storage switch (WSSW). Convert contents of DATSW to disk blocks. If DATSW is zero then exit to SC15. Put contents of DATSW into COM69 (disk block count of program in Working Storage). Exit to SC15.

SC15 Put contents of COM69 into COM54 (disk block count of program being stored). Exit to NYET4 if zero disk block to be stored. Exit to SC17.

FRPT Set paper tape switch. Go to RIOCS to bring in required paper tape I/O routine, and return. Exit to FRIN.

FRIN Set principal I/O switch. Clear disk block count of program in Working Storage (COM69). Exit to SC17.

SC17 Scan the TO field for Working Storage, Fixed Area, or User Area indication. Check columns 17 and 18 and exit to:
SC17A if not Working Storage.
NYET5 if Working Storage indication in both FROM and TO fields.
Set Working Storage switch. Exit to VALST.

SC17A Check columns 17 and 18 and exit to:
SC17B if neither Working Storage or Fixed Area. Set Fixed Area switch. Exit to NYET5 if no Fixed Area defined. Go to CKTMP and return if not temporary mode. Exit to SC21.

SC17B Exit to NYET5 if not User Area (invalid TO field). Set UASW in DUPCO. Exit to SC21.

SC21 Go to SNAME to convert and store name into COM51 and COM53 of COMMA. Go to LETSR to do a LET search and return. Exit to VALST if name was not found or if name was found and mod switch has been set; otherwise, restore the disk block count. Exit to NYET6 (invalid NAME field).

Decode DUMP Function

This routine decodes the balance of the function field, columns 7 through 12, as well as the FROM field, the TO field, the NAME field, and the COUNT field, as required.

DACTL If balance of name data is not in the function field, exit to NYET3; otherwise, exit to DACNT to set DATSW with the actual sector count as punched in the card. Convert DATSW value to disk block count and store in DATSW. Exit to NYET7 if COUNT field was invalid. Put contents of DATSW into COM54 (disk block count of program to be dumped). Exit to DPCTL.

DPCTL Scan columns 13 and 14 and exit to:
FRWS if from Working Storage.
FRUA if from User Area.
NYET4 if invalid FROM field.
Set FXSW in DUPCO. Exit to SCN17.

FRUA Set UASW in DUPCO. Exit to SCN17.

FRWS Convert WS/SAD to disk blocks. Set WSSW in DUPCO. Set disk block address of program to Working Storage address. Exit to SCN17 if Dumping Data. Put disk block count of program in Working Storage into disk block count of program to be dumped. Exit to SCN17.

SCN17 Scan the TO field for WS. If present, set WS switch negative. Exit to DNAME. If WS not present, set I/O switch. If PR present, set principal print switch and restore page switch. Skip 1 line. Exit to DUDA if from Working Storage to printer indicated; otherwise, exit to DNAME. Exit to DNAME if CD in columns 17 and 18 and card is principal input device. If CD in TO field, exit to DNAME. Exit to TOPT if PT in columns 17 and 18 or CD in same columns and PT is principal input. Exit to NYET5 if invalid TO field.

TOPT Set paper tape switch. Go to RIOCS to get the required paper tape I/O routine from disk to core and return. Exit to DNAME.

DNAME Go to SNAME to convert and store the name in name code (5 characters in 2 words) in COM51 and COM53 of COMMA. If dumping from Working Storage, then exit to VALDU; otherwise, go to LETSR to search LET for a name in LET. If name was not found, exit to NYET6 (invalid NAME field). If dumping to printer, then exit to DUDA; otherwise, exit to DUUA.

DUDA If data format has been forced, then enter COM54 (disk block length of program) into DATSW; otherwise, insert contents of DATSW into COM54 (disk block length of program). In either case, if dumping from Working Storage, exit to VALDU. Exit to DUUA.

DUUA Set up XR2 to permit entering dump program at DWSC + 2. Exit to PLUS2.

VALDU Adjust XR2 to enter dump routine at DWSC + 3. Exit to PLUSX.

Decode DELETE and DEFINE Functions

DLCTL If function field not DELETE, then exit to NYET3 (improper function field). Go to SNAME to convert and store name in name code in COM51 and COM53 of COMMA. Go to LETSR to Search LET for the name and return. If name was not found, then exit to NYET6 (invalid NAME field). Exit to CDEL to call DELETE from disk.

DFCTL If DEFINE FIXED AREA in function field, exit to DACNT to set the cylinder count from the control record into DATSW. Return. Exit to DFXA2, (call DEFINE from disk). If DEFINE VOID FORTRAN is in function field, exit to DVFOR, (call DEFINE from disk). If DEFINE VOID ASSEMBLER is in the function field, exit to DVASM, (to call DEFINE from disk). If none of the three above, exit to NYET3 (improper function field).

Call Required DUP Function

RIOCS A closed subroutine to read the paper tape I/O routine from disk to core. The

paper tape routine will overlay the card I/O routine in core. Address of calling routine is in RIOCS. Exit to calling routine.

VALST — XR1 is set to the entry point address of the store routine. XR2 is set for the size of the LET entry. If the program is to be stored from an I/O device, exit to TFILE. Exit to NOIO if Storing Data. Go to GET for first eleven words of program from Working Storage. Exit to SEXIT if TYPE field zero. Exit to NOIO if NAME Field blank. Exit to SEXIT if not same NAME. Exit to NOIO.

NOIO — Exit to CICAL if Storing Core Image. Exit to DACAL if Storing Data. Exit to CICAL if Storing to Fixed Area without specifying CI or Data. Exit to DACAL.

DACAL — Adjust entry point into Store by +4 for I/O not required. Exit to CALST.

CALST — Save LET entry length in LREQD. Insert required entry address of STORE block into GET, and set XR3 to point at the IOAR Header address of STORE block. Exit to GET+1 to get STORE block from disk.

CICAL — Adjust entry point into Store for I/O required and CI. Exit to CFILQ.

TFILE — Exit to CALST if not CI. Exit to CFILQ.

CFILQ — Insert entry point in FILEQ routine into GET. Exit to GET to call FILEQ from disk to core.

FILEQ — Subroutine. Called by DCTL when storing a mainline program in Core Image format. Entry Point is CICAL.

CICAL — Set up COM52 and COM48 (in COMMA) to specify source of DSF program to be converted to Core Image and force a core map to be printed by the Loader. Initialize Supervisor Control Record Area of disk to indicate no Supervisor control records. Go to FILES, if *FILES records are to follow *STORECI DUP control record, and return. Exit to GET + 1, to read the STORE Function from disk to core and enter at location specified by STSW (in DUPCO). This will be either WSD (I/O required) or WSD + 2 (I/O not required).

FILES — Uses a subroutine (INPUT) to blank the input buffer, read a record and print that record. Decoding will then proceed giving error messages if record is not properly identifiable as to type, character sequence, or if too many files are defined. The number of the file is decoded and converted to binary. The name is decoded and stored in name code. Scan continues until all of the card has been scanned. If another record is required, it is read and printed and scanning is repeated until record count (in COUNT field of STORECI DUP control record) is satisfied. The accumulated output is written on the fifth sector of the Supervisor Control Record Area. Return to calling routine.

LECTL — If function field not DUMPLET, exit to NYET3 (invalid function name).

LET2 — Set DUMPLET switch, principal print switch and I/O required switch. Set XR2 with the IOAR Header address of DUMPLET. Exit to PLUS2.

CDEL — Go to CKTMP and return if the Monitor is not in temporary mode. Set XR2 to IOAR Header for DELETE. Exit to PLUS2.

DVASM — Set VOID ASSEMBLER switch in DUPCO. Exit to DFOUT.

DVFOR — Set VOID FORTRAN switch in DUPCO. Exit to DFOUT.

DFXA2 — Convert cylinder count that is in DATSW to disk block count and store in Fixed switch (FXSW) in DUPCO. Exit to DFOUT.

DFOUT — Go to CKTMP and return if the Monitor is not in temporary mode. Load XR2 with IOAR Header address of DEFINE block. Exit to PLUS2 to call in DEFINE block.

VASTM — Go to CKTMP and return if the Monitor is in temporary mode.

CALLE — Load XR2 with IOAR Header of EDIT block. Exit to PLUS2 to get EDIT block from disk.

WACTL — Go to CKTMP and return if not in temporary mode. Exit to NYET3 if DWADR not in function field. Load XR2 with IOAR Header of DWADR function. Exit to PLUS2.

PLUS2 — Save XR2 (IOAR Header address). Adjust entry address by +2. Exit to PLUSX.

PLUSX — Store required entry address to GET. LOAD XR3 with required IOAR Header address. Exit to GET+1 to get required DUP function.

CKTMP — Subroutine to check operational mode of the Monitor. Exit to NYETT if in temporary mode. Return to calling routine if not JOBT status.

Decode DUP Control Record COUNT Field

DACNT — A closed subroutine to read the decimal value from columns 27 through 30 of the control record and enter it into DATSW in hexadecimal. If non-numeric character encountered, exit to NYET7 (invalid COUNT field). XR1 and XR3 are used as they are and are not altered. XR1 is pointing to columns 3 and 4 of the control record and XR3 is pointing to L in DUPCO. Address of calling routine is in DACNT.

Convert NAME to Name Code

SNAME — Convert name from packed EBCDIC to name code. The converted name will be stored in COM51 and COM53 of COMMA. If the first character of the name is numeric, exit to NYET6 (invalid NAME field). XR2 is saved and restored. While in the routine, XR2 points to COMMA. XR1 points to card image columns 3 and 4 and is left unmoved. Address of calling routine is in SNAME. After conversion is complete, exit to calling routine.

LETSR — A closed routine that will compare a name in COM51 and COM53 of COMMA with each LET entry in turn until a match or end of LET has been found. The search of LET will always include a search of FLET if a Fixed Area has been defined. If the name is found, the first three words of LET entry are stored in COMMA and the disk block address of the program is stored in PGMAD. The name switch (NAMSW) is also set. If the name was not found then the disk block address of the last program stored is available in COMMA (COM12). If COM12 does not agree with the calculated disk block address (COM52), then exit to SYSCK. SAD of the last LET sector is inserted into LETSA. The program disk address is stored in PGMAD. The word prior to the LET entry is stored in the delete switch. XR2 and XR3 are saved. XR2 is used to point to COMMA, while XR3 is used to point to LET entries minus 1. Address of the calling routine is in LETSR. When the LET search is completed, exit to the calling routine.

DUMP Function

Chart: BD

- Moves programs and data from the User Area or Fixed Area to Working Storage.
- Dumps programs and data from Working Storage to the principal I/O device.
- Performs required data format conversions.

The DUMP function is comprised of two basic subfunctions. One subfunction obtains a Data file or program in Core Image or Disk System format from the User or Fixed Area and moves it to Working Storage.

The other subfunction, using the DUP I/O routines, outputs the data or program from Working Storage to the specified I/O device. This subfunction converts the program or data in Working Storage from the format of the FROM field to the format of the TO field. The individual I/O routines convert the DUMPed information to the character codes appropriate to the devices they service.

The operation of the DUMP function is entirely controlled by switches and parameters in DUPCO, set up by DCTL according to the DUP control record (*DUMP) and the LET search. The contents of the FROM and TO fields in the control record determine whether only one of the two or both of the DUMP subfunctions are to be executed.

Entry Points

DWSC+2 — Entry point for dumping programs or data from User/Fixed areas to Working Storage. Exit to DWS.

DWSC+3 — Entry point for dumping programs or data from Working Storage to I/O device. Exit to WSIO.

Dump User or Fixed Area to Working Storage

DWS — Entry point from DUP Control. Calculates parameter required to GET first block of program from disk. Calculates

parameter required for first block to be put on disk. Calculates parameters required to check for last block. Checks disk blocks of program not yet moved. Reduces size of block if too large. Sets ENDSW if balance is to be moved. Gets required block from User Area. Transfers information from Program Header for COMMA. Puts block to Working Storage. Calculates parameters for next block to be moved. When the last block has been moved, sets Working Storage Indicator, COM69, to indicate that Working Storage is occupied. If I/O output is not required, exit to REST in DUPCO; otherwise exit to WSIO.

DUMP Working Storage to Principal I/O Device

WSIO — XR3 points to disk sector buffer. XR2 points to the Communication Area (COMMA). If principal print device required, exit to PRPDF; otherwise, get first block from disk. Initialize the sequence number and, if a Core Image program or data to be dumped, exit to CDDF. Process Program Header and set up DFPNT point at the second data header word. During the dump from Working Storage to I/O, the DFPNT will always point in front of an indicator word. The data header will be saved in CTL and CTL+1. Fill in the Program Header with required information from COMMA; the effective length of the program, the length of the program (in disk blocks required), the name of the program, and, if word 12 of the program header is 0, fill in the execution address from COM56. Otherwise fill in COM56 with word 12 of the Program Header. Decode the type field (word 3) of the Program Header, and using the branch sector in TYPV, branch to the proper routine to handle the particular type of program that is being dumped. This provides for the punching of the C-type records in front of the header and in back of the header.

NOTE: In its distributed form, the DUMP function will not output the Loader overlay and Loader restore records of an 1130 Card/Paper Tape System program which has been stored on the disk with these records included.

However, if the user wishes to output these records, the following change must be made to the DUMP function:

This pair of instructions

```
TYP3    MDX         ML2
        BSC     L   SPGMX
```

must be replaced by this pair of instructions

```
TYP3    STO         CSW
        LDX     L3  /0E00
```

This change will permit the outputting of the Loader overlay and restore records. If this change is made the following holds true.

The various C-type Loader overlay cards have been recorded and are stored on disk in the DUP working block. For type 3 or 4 programs, three Loader overlay records will be punched prior to the header record followed by three Loader restore records. For type 5 or 6, four Loader overlay records will be punched prior to the header, then the header, and then the Loader restore records. For type 7, there will be one Loader overlay card punched prior to the header record, followed by the header record and the three Loader restore records. If the program is a type 1 or type 2 then the Program Header card will be punched out without any Loader overlay records or Loader restore records. Exit to SPGMX.

MOVE — Move the number of words specified by MCNT from the location specified by XR3 to RD1 buffer. Zero the header check sum word. Exit to PCHOT to punch the record and return. Address of calling routine is in MOVE. Exit to calling routine. XR2 is left set at RD1-1.

SPGMX — Sets XR2 to RD1-1 (start of output). Exit to CONV.

CONV — Converts Disk System format data words to Card System format data words, includes the proper indicator words and forms the new data headers as required. If end of program data header, exit to EOP. Load XR1 with the contents of DFPNT. This now points at word count field of next data header. If sufficient words to require only one card, then exit to SMALL, otherwise, exit to CLEAR to clear 60 word buffer at RD1 and return. Decrement CTL by 45 data words. Set CNTI equal to 5. Increment

cycle by 1 for type and count of data words and store in word 3 of card image. Exit to DATCD.

CLEAR — A closed subroutine that clears 60 word buffer at RD1 to zeros. Uses XR2 and XR3. Saves and restores both index registers after 60 words are cleared. The first word of card image is filled from CTL which contains the core location of the first data word on the card. The A and Q registers are cleared. The calling routine address is in CLEAR. Exit to the calling routine.

DATCD — Moves indicator bits and data words from Disk System format to Card System format based on an eight-card cycle. The eight-card cycle may be broken into three basic subdivisions. The first case is when the last indicator bits saved were from cards 1, 2, 4, 5 or 7; the second case is when the last bits saved were from cards 3 or 6; and the third case is if the last card cycle was number 8 where no bits had to be saved. Indicator bits are saved in SVIND and are shifted appropriately on each card cycle. The indicator bits are also checked to see if LIBF subroutines are indicated; if so, the core load address is decremented for the next data header. The movement of the data words from Disk System format to Card System format is also based on the eight-card cycle, with the three same subdivisions. The data words saved from the previous block process are moved into the card image. A nominal 40 words are then moved from the Disk System format to Card System format buffer. The data words are considered in blocks of 8 and any data words that are not needed to fill the card will be saved, in DWRD1. Exit to BUMP.

SUBT9 — Decrement DFPNT by 9. Reset card 3 or 6 switch. Exit to BUMPA.

BUMP — If second card or fifth card completed, exit to SET36. If third or sixth card completed, exit to SUBT9. Exit to BUMPA.

BUMPA — Increment DFPNT by 54. If remaining data word count is not more than 3, then decrement DFPNT by 1. Exit to PCHOT to punch a record and return. Exit to SPGMX.

SET36 — Set card 3 or 6 switch. Exit to BUMPA.

PCHOT — Closed subroutine to punch out a data card. XR2 and XR3 saved and restored. Exit to CHKCD to check for blank record and return. Exit to CDPCH to punch a card or paper tape record. If DFPNT has moved past the sector boundary of the Disk System format, then decrement DFPNT by 320. Exit to GETX to get next sector from Working Storage and return. Restore XR2 and XR3. Address of calling routine is in PCHOT. Exit to calling routine.

Blank Card/Monitor Control Record Test

CHKCD — Closed subroutine with calling routine address at CHKCD. If paper tape, return to address at CHKCD. Exit to GETCD to read a card and return with card converted to packed EBCDIC in CRBUF-40. If non-blank, exit to NONBL. After 80 columns checked, exit to calling routine.

NONBL — Exit to SEXIT to bring in error print package and return. Continue to RDRDY.

RDRDY — Check to see if reader ready. Loop until it is; then return to CHKCD+1 to test next card for blanks.

Building One Complete Card or Less

SMALL — Number of data words left is insufficient to fill more than one full card; therefore, DFPNT is preceding data header by the count in DATAC. If words required to complete card image are less than those saved, exit to MVEPT. Calculate number of indicator words and number of data words actually left available for this card image. Move DFPNT to end of next data header. Exit to CLEAR to clear 60 words of buffer at RD1 and return. Save new data header in CTL and CTL+1. Fill in word 3 of card image with type and word count. If more than the saved words are available, exit to BRARD. Enter indicator bits from SVIND to card image. Enter data words from DWRD1 to card image. Exit to CLRCY.

MVEPT — Adjust count of saved data words to 2. Exit to ZROWD.

BRARD — Handles cases where words saved do not extend to the data header and therefore more words must be moved into the card

image, likewise more indicator bits. Put indicator bits from SVIND and from Disk System format into card image. Move data words left over from DWRD1 into front end of card image. Move remaining Disk System format data words preceding new data header to the card image. Clear CYCLE and card 3 or 6 switch. Exit to BUMP2.

GETX — Sets XR3 to point at IOAR Header for buffer. Increments sector address by 1. Exit to GET to read indicated words from disk to buffer indicated by XR3 and return. Address of calling routine in GETX. Exit to calling routine.

EOP — Set XR2 to point at RD1-1. Exit to CLEAR to clear a 60 word buffer at RD1 and return. Fill first word of end of program card from first word of end of program data header (effective length of program). Insert into word 3 of card image at the end of program type. Obtain the execution address from COM56, and insert into word 4 of card image. Exit to PCHOT for punching records and return. Exit to REST (DUMP complete). Return to DUPCO (print exit message).

Format Data Input for Punching

CDDF — Obtain disk block count of program from DATSW. If disk block count zero, branch to GOGO.

CDDFI — Exit to DCNT to convert disk block count to a word count and return.

NEWCD — If the word count is equal to or less than 54 words, exit to FFAFF; otherwise, set XR2 to point to RD1-1. Exit to MOVE to move 54 words, punch a record, and return. If sector of data completed, exit to GETY; otherwise return to NEWCD.

FFAFF — If ENDOJ not equal to zero, and last card punched, exit to REST. If ENDOJ zero, return to CDDF1. DUMP complete return to DUPCO; otherwise, reset XR1 to RD1-1, exit to CLEAR and return. Exit to MOVE and return.

GOGO — Exit to REST (DUPCO).

DCNT — A closed subroutine that supplies CDDF routine with a maximum of 2160 words to punch.

PRPDF — If zero word count, exit to REST. Insert count of 320 into IOAR Header for getting sector from disk. Exit to GET to get specified count from disk. IOAR Header core location is specified by XR3. Convert disk block count of program to word count and insert in count field of print binary call. Set XR3 to IOAR Header for principal print I/O routine. Exit to GET to get the required words from disk and return. Exit to PRTBI to print the entire program in binary as specified in word four of the calling sequence and return. Exit to REST. When the DUMP to the printer is complete, give control to DUPCO to print out exit message.

DELETE Function

Chart: BE, BF

- DELETEs programs from either the User or Fixed Area.
- Updates and packs LET/FLET if required.
- Packs programs in the User Area.
- Updates DCOM (on disk).
- Updates COMMA (in core).
- Provides a System Check if DCOM and COMMA do not agree.

DELETE Program From User Area
The program is divided into two phases:

Phase I packs LET by the number of words made available by the deleted LET entry. This packing will take place in the sector containing the entry to be deleted and all subsequent sectors, except that six-word entries or multiple entries are not split across a sector boundary. Only when room for a complete entry exists will that entry be moved across a sector boundary. The LET sector that contains the program entry to be deleted is assumed to be in core and DELSW is assumed to be pointing to the first word prior to the LET entry that is to be deleted. PGMAD is assumed to contain the disk block address of the program.

Phase I also calculates and stores the necessary parameters for Phase II.

Phase II packs the programs into the User Area by the size of the deleted program. Disk System format programs are moved by an exact number of disk blocks until the first Core Image program or Data file is encountered.

Then the programs are packed only to the nearest sector boundary since Core Image programs and Data files must start at a sector boundary. A program in Working Storage will also be moved so that it still begins at the start of Working Storage, if this boundary moves.

DCOM (on disk) and COMMA (in core) are updated. Control is returned to DUPCO which initializes and causes DCTL to print out the exit message.

Entry Points

There is only one entry point, DELC+2, which is the first word of the first sector of DELETE. Control is passed to DELETE from DCTL after reading a DELETE control record and finding the program entry in LET/FLET. This sector of LET/FLET is left in core for use by DELETE.

Phase I - Packs LET

DEL00 Exit to SEXIT if Working Storage will be exceeded when moving programs. Exit to WSOK if Working Storage not exceeded.

WSOK Exit to DEL01 if program to be deleted is in User Area. Turn on FXSW. Exit to DEL01.

DEL01 Use XR1 to point to first word of LET entry to be deleted. Use XR2 to point to delete constants. Compute number of entries yet to check in this LET sector and store the number in CNTE. Exit to DFL00 if program to be deleted is in Fixed Area. If name specified is other than the prime entry, then scan backwards until prime entry is located. Obtain number of disk blocks to be deleted, including padding, and store in DELDB. Compute new User Area disk block address, and store in NEWUA. Exit to DEL04 if three-word entry. Calculate actual disk block count program required and the new padding required for the Core Image program or Data file. Exit to DEL04.

DEL04 Go to DEL05 and return. Exit to DEL18.

DEL05 A closed subroutine to scan for multiple entries being deleted to determine the number of words to be shifted left in LET. Exit to DEL09 if last entry of sector is encountered during search; otherwise, after first LET entry that is not to be deleted has been located, go to DEL10 to scan LET for the first Core Image program entry that follows. Record the pertinent information and return. Exit to DEL09.

DEL09 Record the disk block count until the next Core Image program or Data file has been encountered. If there is no Core Image program or Data file following, return to calling routine. Record the first available address for a Disk System format program, Core Image program, or Data file that may follow the deleted program. Calculate the adjusted amount of padding that would occur before a Core Image program or Data file. Return to calling routine.

DEL10 Closed subroutine to search for Core Image program entry in LET sector. Calling routine address in DEL10. Exit to calling routine if Core Image program or Data file previously located. If next entry is not Core Image program then add the disk block count of program to the accumulated disk block count. Exit to calling routine. If the six-word LET entry is located, set the Core Image switch (CISW) to indicate a Core Image program or Data file. Go to DEL16 to find the actual disk block count required by the program and return. Compute and record sectors by which the User Area must be adjusted. Compute and record new User Area disk block address. Compute and record the new total disk block count for the Core Image program or Data file. Exit to calling routine.

DEL16 Closed subroutine to calculate the actual disk block count used by the Core Image program or Data file. Address of the calling routine is in DEL16. If Core Image format program, the number of disk blocks required equals the word count of the Core Image program divided by 20. For a data file the actual disk block required is in word 6 of the LET entry. Record the actual disk blocks required in DBPGM. Exit to calling routine.

DEL18 Set up to move required LET entries from next sector. If last LET sector, set ENDSW and exit to DEL32.

DEL20 Get next LET sector from disk and put into core starting 4 words after end of first LET sector. Set XR1 at first word of LET header in current LET sector.

DEL22, DEL30 — Move LET entries from Sector N+1 to sector N until number of words in sector N is less than the size of the next entry to be moved in from sector N+1. Six-word LET entries and multiple entry programs may not be split over sectors.

DEL32 — Prepare LET sector for writing. Adjust words in header of LET sector N. Move entries required across the sector boundaries from sector N+1 to sector N. Exit to DEL41 (write this sector).

DEL38 — Initialize to process next sector, including the exchange of LET buffers. NOTE: LET buffers are exchanged by merely reversing the addresses of buffer 1 and 2 with a Load Double (LDD), Rotate (RTE) 16, and Store Double (STD). The symbolic references to the addresses are THIS and NEXT. If a Core Image program was encountered, exit to DEL18; otherwise restore XR1 and XR2. Exit to DEL08 (only systems format).

DEL39 — Set LMISW to zero since the last sector of LET is not deleted.

DEL41 — Write disk routine. If a LET sector is deleted, then rewrite the needed LET sector header. If all programs in User Area have been deleted, exit to DEL39. If the previous LET sector not already in core, read previous LET sector and update header. Exit to DEL42.

DEL42 — Update header of previous LET sector.

DEL43 — Calculate number of words to write and sum the disk blocks referenced in this sector. Put the previous sector to disk using PUT.

DEL44 — Exit to DEL38 if more updating of LET is required. Exit to DEL48 if end of LET. Exit to DEL47 if Core Image program or Data file already encountered. Exit to DEL46.

DEL46 — Go to DEL05 to search for Core Image program or data file and return. If Core Image program or Data file processed, then exit to DEL47; otherwise, read next LET sector and exit to DEL46.

DEL47 — Write updated LET sector.

DEL48 — Exit to DEL50 if LET is not shorter. Adjust number of words in LET.

DEL50 — Exit to DFL60 if deletion from Fixed Area; otherwise, exit to DEL60 (go to Phase II, Phase I has been completed).

Phase II - Packs User Area and Updates DCOM and COMMA.

DEL60 — Set up parameters for MOVER subroutine to move programs by disk blocks. Go to MOVER and return. Exit to DEL76 if any programs to move by sectors. Exit to DEL86.

DEL76 — Set up parameters for MOVER subroutine to move programs by sectors. Go to MOVER and return. Exit to DEL86.

DEL86 — Get Disk Communication sector (DCOM) from disk. Set XR1 to point at Core Communication Area (COMMA). Set XR2 to point at delete parameters. Set XR3 to point at Disk Communication Area just brought into core. Update COM4 (word address of next LET entry) both in COMMA and DCOM. Exit to SYSCK (system check) if the two are not equal. Adjust COM10 (word address of next LET entry adjusted) both for COMMA and DCOM. Exit to SYSCK (system check) if both do not agree. Update COM6 (first available disk block address in User Area base) of both COMMA and DCOM. Exit to SYSCK (system check) if they do not agree. Update COM12 (next available disk block address of User Area adjusted) in both COMMA and DCOM. Exit to SYSCK (system check) if both are not the same. Update file protect address in both DCOM and COMMA. Exit to SYSCK (system check) if the DCOM and COMMA do not agree.

DEL88 — Update FPA and FPAD (File Protect Address Base and Adjusted); i.e., the beginning of Working Storage on one disk. Rewrite DCOM back to disk. Exit to DEL90.

DEL90 — Exit to REST (return to DUPCO).

DELETE Program From the Fixed Area

Phase I replaces the program entry by a dummy entry. If there is an adjacent dummy entry, the dummy entries are combined to form a single dummy

entry and FLET is shrunk by 6 or 12 words depending whether there were 1 or 2 adjacent dummy entries. Phase II - The programs in the fixed area are not moved. Only DCOM and COMMA are updated and control is returned to DCTL which reads the next control record. Phase I - updates FLET.

DFL00 Convert FLET program entry to dummy entry. Exit to DFL30 if not first entry in sector. Exit to DFL40 if first FLET sector. Read previous FLET sector. Exit to DFL40 if first entry on previous sector is not a dummy entry. Combine dummy entries. Exit to DFL35.

DFL30 Exit to DFL40 if previous FLET entry not a dummy. Combine dummy entries.

DFL35 Increment SHR by 6 (number of words to shrink FLET).

DFL40 Exit to DFL05 if not last entry in sector. Exit to DFL20 if last entry in FLET. Exit to SYSCK (system check) if next FLET sector address is missing from this sector header. Read next FLET sector. Exit to DFL20 if first entry is not a dummy entry. Combine dummy entries. Exit to DFL15.

DFL05 Exit to DFL20 if next FLET entry is not a dummy entry. Combine following dummy with this dummy entry.

DFL15 Increment SHR by 6 (number of words to shrink FLET).

DFL20 Exit to DFL50 if FLET not condensed. Shrink FLET by SHR words. This is accomplished by using the LET updating part of Phase I for deleting routines (DEL18 thru DEL50) from the User Area. The following parameters are set for that purpose: DEL34+1, TO, CISW, CNTE. Go to DEL18 to shrink FLET by SHR words and return to DFL60.

DFL50 Rewrite uncondensed, but updated, FLET sector.

Phase II - Updates DCOM and COMMA.

DFL60 Read DCOM from disk. Update COM5 in DCOM and COMMA by the number of words in FLET. Exit to DEL88 to write DCOM back on to the disk.

Exit

If no system checks occur then DELETE exits to DUPCO which initializes and causes DCTL to print the exit message.

STORE Function

Chart: BG

- Stores card input in Working Storage, including any required data format conversion.
- Transfers control to the Loader, which converts to Core Image format and returns control to STORE.
- Updates LET/FLET as required.
- Updates DCOM as required.
- Calls in the Loader to convert DSF programs to CI format when STORE CI has control.
- Stores Working Storage in either the User or Fixed Area, including any required data format conversion.

The STORE function is comprised of two basic subfunctions. One subfunction, using the DUP I/O routines, inputs a Data file or program from a specified I/O device to Working Storage. The I/O routines convert the information to be STORED from the character codes of the I/O devices to packed EBCDIC. This subfunction converts the program or data from the format of the FROM field to the format of the TO field.

The other subfunction moves a program or Data file from Working Storage to the User or Fixed Areas. Upon completion of the STORE, this subfunction updates LET/FLET and DCOM to reflect changes to the User and Fixed Areas.

The operation of the STORE function is entirely controlled by switches and parameters set up by DCTL according to the DUP control record (*STORE, *STORECI, or *STOREDATA) and the LET/FLET search. The contents of the FROM and TO fields in the control record determine whether only one of the two or both of the STORE subfunctions are to be executed.

The Working Storage to User/Fixed Area routines precede the I/O to Working Storage routines in core so that they may expand and use part of the I/O to Working Storage routine area as a buffer.

Entry Points

WSD Entry point from DCTL function when I/O is required; i.e., Card or Paper Tape to Work Storage, User Area, or Fixed Area. If storing in Core Image

format to either User Area or Fixed Area then this entry is from DCTL function through the called FILEQ routine. Exit is to IOWS.

WSD+2 Entry point from DCTL function through the called FILEQ routine when Storing Core Image and I/O is not required. Entry point from IOWS (STORE function) if Core Image and I/O was required. Exit is to CICAL.

WSDP+4 Entry point from DCTL function when not Storing Core Image and I/O is not required. Entry point from IOWS (STORE function) if not Core Image and I/O was required. Entry point from Loader thru DUPCO when conversion to Core Image has been completed.

Card System Format

Records are read, check summed, and examined for:

Loader Overlays-Checks length of header and creates Program Header with required words saved in COMMA.

Data Records -Indicator bits and data words are converted to Disk System format with data headers inserted as required.

F-type Record -Terminates reading from I/O device. Furnishes information for Program Header and last (EOP) data header; forces transfer of Disk System format program from core to Working Storage.

NOTE: If core buffer fills before F-type record is read, the buffer is transferred from core to Working Storage and then the reading and conversion of records continues.

Card Data Format

Records of 54 words are read and transferred to core in sequence until the number of records specified on the DUP control record are read or until the buffer in core is filled. In either case, the buffer is then written to Working Storage without any conversion. This process continues until the record count has been satisfied.

Termination for Both

Return control to DUPCO for initialization and call of DCTL to print exit message if Storing to Working Storage only. Exit to WSD+2 if Storing Core Image. Exit to WSD+4 if Storing Data or Disk System format.

Card to Working Storage Routines

Records are read from the I/O device in Card System or Card Data Format.

Assume that LET has been searched for name and, if name found, then all words of the LET entry have been saved and name switch is set. If no name found, then sector and word addresses where an entry may be made are made available.

Multiple entry point names will be checked when going from Working Storage to User/Fixed Area.

Working Storage to User/Fixed Area

In general, this picks up the program from Working Storage and transfers it to either the User Area or the Fixed Area, filling in program headers from COMMA.

If the program is a multi-entry subroutine, then a LET search is done for each of the secondary entry points. This is accomplished by calling an overlay (SRLET) from disk.

The program may be in Working Storage either from a previous STORE from I/O to Working Storage, a compilation, an assembly, or a DUMP from User Area to Work Storage. Parameters generated by the FORTRAN compiler or the Assembler, after the header has been written, are written by them into COMMA. It is these parameters that are selected from COMMA and placed into the program header as it is stored from Working Storage to disk, User or Fixed Area.

NOTE: If Storing to the User Area, the program in Working Storage is partially overlayed.

As the program is moved from Working Storage to the User Area, LET and COMMA are adjusted in the following sequence:

1. GET X-1 (nominal) sectors from Working Storage where X is the number of sectors in Disk System format buffer.
2. Get the sector containing the start of the last program in User Area.
3. Fill in program header from COMMA.
4. Put X sectors into the User Area.
5. Add program name and length to LET.
6. Adjust COMMA and DCOM.
7. Exit to REST in DUPCO function.

IOWS

Enter from DUP Control. Record file protect address in working constant M. Compute buffer size required for IOWS to allow one more card than integral number of 320-word sectors, and put word count of IOWS buffer into L2. Exit to DAFMT if storing data. Exit to CDGET to read binary and pack into CD+1 through CD+54. Set XR2 to point to DF+1; i.e., the disk buffer. Go to MOV54 to move 51 words from card buffer to disk buffer, and return. This saves the maximum length of Program Header. Set XR1 to DF+13 (the position of the disk data header for mainline programs). Decode type of header card. Calculate displacement for branch instruction that permits adjustments required by each header type. If type 3 or 4, then Program Header length is 9 plus number of entry points. If the program is not type 1 or 2, then there may be some Loader overlay cards preceding the header and some Loader restore cards following the header. If they are present, they will be read and stored on disk in the proper locations to permit punching them out when the program is dumped. Types 3 and 4 have three Loader overlay cards and three Loader restore cards, while types 5 and 6 have four Loader overlay cards and the same three Loader restore cards. Type 7 has one Loader overlay card and the same three Loader restore cards. Loader overlays for types 3 and 4 are placed in the DUP working block at C1D, while the Loader overlay cards for type 5 and 6 are placed in C2D. The Loader overlay cards for type 7 are placed at C3D, and the Loader restore cards are placed at C4D. Each one of these areas is one sector long. When a dump is to be done for these respective types, the proper Loader overlay cards and restore cards could be dumped out in the required sequence coming from this area. The branch table (BR) is used to determine the branch location for the various types. There are 16 words and 16 types. Exit to HDEND, if type 1, mainline, absolute. Exit to HDEND, if type 2, mainline, relocatable. Exit to TY348, if type 3, (Non-ISS LIBF) library function, one word LIBF. Exit to TY348, if it is type 4, (Non-ISS CALL) it is subprogram, two-word call. Exit to TYP56, if it is type 5, one-word LIBF, ISS routine. Exit to TYP56, type 6, two-word call, ISS routine. Exit to TYP7, type 7, is ILS routine. Exit to SEXIT, if type A data is not legitimate at this time. Exit to CTYPE, type C automatic load card. Types D and E are automatic load cards. Exit to AREAD, which in effect ignores them. Exit to SEXIT if type F, end of program card, is invalid at this time. Exit to SEXIT, (print error message) types 0, 8, 9 and B, which are reserved for system expansion.

TY348

Exit to SEXIT if a type C record was omitted (not three type C records read). Set up to write three records to disk on one sector. Exit to NAME.

TYP56

If a type C record was omitted (not four type C records read), exit to SEXIT. Set up to write four type C records to disk in one sector. Exit to NAME.

NAME

Compare name on control record and Program Header. Exit to SEXIT if no match (print NAME error message). Exit to WRTC.

TYP7

If more than one type C record read, then exit to SEXIT (print error message). Set up to store one record in one sector on disk. Exit to WRTC.

WRTC

Go to PUT to write on disk and return. (Type C records read will be written.) Calculate special length for Program Header types 3, 4, 5, 6, and 7. Exit to SEXIT if header length invalid. Exit to READ2.

READ2

Go to CDGET read binary non-header records, pack (at CARD) and return. Set XR1 to point at buffer for Working Storage. Set XR2 to point at packed record input. Exit to SEXIT (Invalid Type) if type not greater than 9. Exit to TEOP if type greater than 9.

TEOP

Branch vector type F exits to EOPCD. Type A exits to DATRC. Type C, D and E exit to TESTC. Type B exits to SEXIT (Invalid Type).

TESTC

If type D or E, exit to READ2. If type C, set XR2 to point to output buffer. Go to MOV54 to move 54 words from card input buffer to locations specified by XR2 and return. Adjust pointer and

	count C-type cards. Exit to READ2 (get another record).
CTYPE	Set XR2 pointing to output buffer. Go to MOV54 to move 54 words from card input buffer to location specified by XR2 and return. Adjust pointer and count of C-type cards. Exit to AREAD (get another record).
MOV54	A closed subroutine to move 54 words from card input buffer to locations specified by XR2. Calling routine address is in MOV54. Set XR1 to count words to move from input buffer. Exit to calling routine.
DATRC	IF mainline program, exit to DATCD. Exit to NONC if no Loader restore records. Exit to SEXIT (Invalid Type) if not 3 Loader restore records. Initialize count and sector address for writing Loader restore records to disk. Go to PUT to write to disk from IOAR specified by XR3 (Loader restore records).
NONC	Turn off C-type switch. Exit to DATCD. Increment Sector Address. Move words in excess of IOWS block size. Exit to calling routine.
LWRIT	A closed subroutine to set COM52 of COMMA to address of Work Storage. Go to PUT to write last part of program to disk and return. Load word count of last part written. Set XR1 to point at COMMA. Return to calling routine.
EOP	Move XEQ address from card image to COM56 in COMMA. Exit to EOP3.
EOP3	Entry point for end-of-program record processing. If non-mainline program, COM56 already contains address of entry point 1. Compute words required to be written in Working Storage and new sector address. Update COM60 with effective length of program, and insert into first word of end-of-program data header. Set second word of end-of-program data header to 0. Go to LWRIT to write last parts of program to disk and return. Convert word count to disk blocks required by program. Insert disk blocks required into COM54 in COMMA, (disk block count of program being stored) and COM59 (disk block count of program in Working Storage).
WRAP	If Working Storage switch not zero, then exit to REST (job complete return to DUPCO). Exit to CICAL if Storing Core Image. Exit to WSDP4 if data or storing to User Area.
CICAL	Put sector of DUPCO to a temporary disk sector (on the DUP working cylinder). Loader will return saved sector to core when Core Image conversion complete. Read one sector of Loader to RLDC. Branch to LOENT. Loader will convert program in Working Storage, in Disk System format, to program in Core Image Buffer, in Core Image, then get DUPCO and enter at RLEPT. Address of RLEPT is in COM44 of COMMA to signal DUP call of Loader. STORE entry address from DUPCO is in CISW. DUPCO will recall STORE to core and exit to the address in CISW.
DAFMT	Assumes data switch contains the record count. Go to CDGET to read and pack binary data records and return. Set XR2 to point at buffer. Go to MOV54 to to move 54 word blocks into disk buffer and return. Increment disk buffer pointer by 54. If buffer is full, go to WRITE to write buffer and return. Decrement card count by 1. If not last card, exit to READD to get another card; otherwise, insert sector address. Calculate word count required. Go to LWRIT to write last part of program to Working Storage and return. Compute number of full sectors. Put disk block count of program into COM54 of COMMA. Put disk block count of program in Working Storage into COM69 of COMMA. Exit to WRAP.
GCIB	Exit to SEXIT if non-DUP switch on. Calculate words of Core Image program now in the CIB and upper core. Exit to GCIB2 if Storing Core Image to Fixed Area. Exit to SEXIT if Core Image program will exceed Working Storage.
GCIB2	Go to GET to read the Core Image header from the first sector of the CIB, and return. Exit to FXA if storing to Fixed Area. Exit to CIFX.
MOVER	Closed subroutine to move words from location specified by XR1 to location specified by XR2 and number of words

specified by XR3. Address of calling routine in MOVER. Exit to calling routine.

CIGO — Calculate core address and word count of last sector written on disk. Go to MOVER to move those words to front of buffer, and return. Go to MOVER to move words in upper core next to words remaining from last sector written, and return. Go to MOVER to move words of Core Image header to output buffer, and return. Go to PUT to write last sector plus excess words in core, and Core Image header to disk, and return. Exit to DOLET.

WSDP4 — Enter from DUP Control, STORE, or DUPCO. Exit to SEXIT if Working Storage will be exceeded by program specified. Initialize by setting WDPGM to program disk block count times 20, that is, the word count. Set WDUA to 16 minus DBADJ all multiplied by 20; i.e., the words previously written in the last sector that will be brought back down to core. CNT is the maximum number of words in the Disk System format buffer.

NOTE: Compute SAD, SADP, CAD, CADP, CNT, CNTP as required for initial GET from Working Storage; also for PUT considering disk blocks already in use and sector containing first disk block of the location where the next program will be stored in the User Area.

CAD is set to Disk System format buffer minus 1.

CADP is set to Working Storage address.

SAD is set to Working Storage address.

SADP is also set to Working Storage address initially.

Exit to GCIB if Storing Core Image.
Exit to FXA if Storing to Fixed Area.
Exit to DOLET if Storing Data file.
Increment CAD by number of words in User Area (WDUA). CNT is set for 3 full sectors (960 words, 3CO Hex Words). SADP is decremented by one sector. Exit to GETWS.

FXA — Compute sector address in Fixed Area and enter into SADP. Compute padding required for FLET entry and record in DBADJ. Move FLET parameters for updating FLET. Exit to SEXIT if insufficient room in Fixed Area for specified program. Exit to CIFX.

GETWS — Adjust CNTP to reflect either the Disk System format buffer size or the words in the program, whichever is smaller. Set WDPGM to zero if last part to be written now. Get block from Working Storage using CAD as the core address of the IOAR header, CNT as the word count, and SAD as the sector address from which to obtain it. Exit to PUTUA if omitting Program Header routine. Set XR2 to point to the words, minus 1, brought down from Working Storage. This permits the operand of instructions using XR2 to refer to the actual word being called. Insert the required words from COMMA into the Program Header just read from Working Storage to core. Exit to LENSR if not type 3 or 4; otherwise, OR the SOCAL class code (type 3 modification) with type 3 and with word count. Exit to LENSR.

LENSR — Set LREQD to the value in word 6 of the header record. This is considered the nominal length of the LET entry. Set PGMHL to LREQD + 9. This is the actual Program Header length. Exit to MULSR if program is type 3 or 4. Exit to SPHA if types 1, 2, 5, 6, or 7.

MULSR — If only one entry point, then exit to SPH; otherwise call store overlay closed subroutine (SRLET) which overlays a portion of IOWS. Exit to SRLET (Search LET) to check that multiple entry points of this program do not already appear in LET, then return. If none of the entry point names have been found in LET or FLET, exit to SPH.

SRLET — Closed subroutine called from disk by STORE, and entered from STORE when storing a subprogram with multiple entry points. Go to GET to read each sector of LET and FLET from disk and return. Compare secondary entry points with each name in LET and FLET.

	Exit to calling routine if secondary names not found in LET or FLET. Print error message with name of entry point found in LET or FLET. Exit to Rest + 1 in DUPCO.
SPHA	Set the size of LET entry, which is 3, into LREQD. Exit to SPH.
SPH	Save Program Header at location of IOWS to permit updating LET at a later time. Exit to GETUA when last sector of User Area still has some disk blocks that may be used; otherwise, the move of the program is not required as it is already located where it will be stored. Put just the header sector on disk by setting CNTP equal to 320 words, CADP equal to CAD, and SADP equal to Working Storage address. Exit to PUTUA.
CIFX	Common exit for both CI and Fixed Area routines. Set header switch to skip Program Header routine. Indicate zero words required from User Area. Exit to GETWS.
GETUA	Get the last partial sector from the User Area and put it into core at the front end of the Disk System format buffer. This data brought in from the User Area does not overlay any of the program which was previously brought in from Working Storage. Exit to PUTUA.
PUTUA	Put block to User Area or Fixed Area. This utilizes CADP for the core address of the IOAR header, CNTP for the word count to be put to disk, and SADP for the sector address to be written to disk. Exit to NDCK if all words of the program have been written. Exit to CI50, if Storing Core Image or Storing to Fixed Area. Step the sector address (SAD) by two sectors. Increment SADP by the same sector count written. Adjust CADP 320 words (1 sector) higher. Zero words in User Area (WDUA). Set CNTP equal to two sectors. Exit to GETWS.
NDCK	Exit to CI60 if Storing Core Image. Exit to DOLET.
CI50	Step SAD and SADP by 4 sectors. This permits storing of data or Core Image programs in 4 sector parts. Exit to GETWS.
GFLET	A closed subroutine to get the current LET or FLET sector from disk. Calling routine address is in GFLET. After the sector is completely on disk, return to calling routine.
NEWHD	A closed subroutine to adjust last LET or FLET sector header, setting the second word non-zero and the fifth word LET sector address plus 1. This routine also writes the last LET sector to the disk. The calling routine address is in NEWHD. NOTE: LET sector header is made up as follows. Word 1 is sector number 0-7 if LET, and 16-23 if FLET. Word 2 is last LET sector or last FLET sector if 0. Word 3 is the disk block referenced by this LET sector. Word 4 contains the number words in the LET sector available for LET. Word 5 contains the sector address of the next LET or FLET sector. This will be zero if this is the last sector to be searched. Exit to calling routine.
DOLET	Go to GFLET to GET required LET or FLET sector and return. Set index to first word of the next new entry. Exit to FXA1 if storing to Fixed Area. Exit to ENT1 if enough words are available in this LET sector to contain the words required for this LET entry (nominally 3 per entry point). Go to NEWHD to adjust header words of LET, write current sector on disk, and initialize next LET sector header, returning when complete.
ENT1	Go to WD123 to insert three-word LET entry and return. Exit to PGM if Disk System format program. Go to WD456 to insert LET entry for Data file and return. Decrement words available in LET sector by 3. Exit to ADJHD (adjust sector header).
WD123	A closed subroutine to insert a three-word entry into LET or FLET. The entry contains the name from COM51 and COM53 in name code and the disk block count from COM54. Restore disk block count into DBADD. Calling routine address is in WD123. Exit to calling routine.
WD456	A closed subroutine to process Data file and Core Image program LET/FLET entries. Modify words 1, 2, 3 as required and insert words 4, 5, and 6 for Data files. In word 1, set first two bits to ones (11) to indicate a Data file. Word 3 is incremented by the number of disk blocks of padding required for sectorization. Words 4 and 5 are reset to zero. Word 6 is the disk block sectorized length of

data in disk blocks. Calling routine address is in WD456. Exit to calling routine.

FXA1 — Sectorize program length and insert to COM54. Exit to MORQD if a new dummy entry is required (disk blocks available are greater than sectorized program disk block length plus padding). Go to WD123 to enter first three words of FLET entry and return. Go to WD456 to enter words 4, 5, and 6 of data program FLET entry and return. Exit to PFXA.

PFXA — Go to PFLET to write FLET sector and return. Exit to FXAL.

MORQD — Go to MVBY6 to move balance of FLET entries by 6 words, so that the new FLET entry may be inserted and return. Go to WD123 to insert first three words of FLET entry and return. Go to WD456 to insert last three words of six-word FLET entry for Data file, modify first three words as required, and return. Reduce the disk block count of dummy entry by the disk block count of the inserted program. Reduce the number of disk blocks referenced in the FLET header word 3, by the number of disk blocks in the entry that was shifted (off the end of disk FLET sector) by MVBY6.

WDATS — Decrement sector header word 4 (words available this sector) by the length of the inserted entry (6 words). Exit to PFXA if more words available in this sector. Exit to NEWFL.

ADJHD — XR1 points to COMMA. XR2 points to word 1 (first available LET entry). XR3 points to IOAR of LET sector. Decrement word 4 of LET sector header (words available this sector) by 3 words. Increment sector header word 3 (disk blocks referenced by this LET sector) by disk blocks of program being stored. Calculate exact word count for this sector and insert into IOAR header. Go to PUT for writing LET sector to disk and return after sector has been completely written. Go to GCOMM to get DCOM from disk, and return after DCOM is in core. Exit to UPDTE.

PFLET — A closed subroutine to write a sector of LET or FLET to the disk with the exact word count required. Exit to SEXIT, (D94), if this entry causes a cylinder overflow of LET or FLET. Calling routine address is in PFLET. Go to PUT (in DUPCO) and return when last word has been written. Exit to calling routine.

MVBY6 — A closed subroutine to shift FLET entries six words, to permit insertion of a new entry. WORDS contains the number of words of this FLET sector which are to be shifted right by 6 words. Calling routine address is in MVBY6. Return to calling routine when move completed.

NEWFL — Insert into FLET header word 4, three words available. Exit to MORFL if not last sector of FLET. Go to NEWHD to make final adjustments of sector header and write it to disk, returning after initializing for next sector. Increment number of words in FLET (COM5) by the size of the next FLET sector header (5 words). Insert leftover (shifted) entry and adjust header. Exit to SAVE6.

MORFL — Go to PFLET to write FLET sector and return. Increment sector address in IOAR header. Go to GFLET to get next FLET sector and return. Exit to SAVE6.

SAVE6 — Save the six words beyond the end of this FLET sector. Go to MVBY6 to shift right all entries in this FLET sector by six words, and return. Insert the six words, previously saved from last FLET sector, into the beginning of this FLET sector. Adjust FLET sector header word 3 by disk blocks of the entry just inserted and subtract the disk blocks of the entry that has been shifted out of this sector. NOTE: If this is a partial sector, then zeros will be in the shifted entry and thus zero disk blocks will be subtracted.

Exit to WDATS.

PGM — XR1 set to Program Header word zero. XR2 points to word 1 (first available LET entry). XR3 set at word 1 of LET entry just filled. Adjust word 3 of the LET entry to reflect the disk blocks required by the Disk System format program being stored. If this program has multiple entries, then enter into

	LET an additional LET entry of three words for each additional entry point. Exit to ADJHD (adjust sector header of LET).
GCOMM	A closed subroutine to get DCOM from disk. Calling routine address in GCOMM. Go to GET, in DUPCO, to read sector from disk. Exit to calling routine.
UPDTE	Exit to PGM2 if neither Core Image program nor Data file. Sectorize disk block address of Core Image program or Data file. Exit to PGM2
PGM2	Record disk block address of program for exit message. Update disk block address of User Area, sector address of Working Storage, and the relative address of the next LET entry, both for COMMA and DCOM. Zero length of program in Working Storage. Exit to PCOMM if in temporary mode. Update the base values.
PCOMM	Go to PUT (in DUPCO) to write adjusted DCOM to disk and return. Exit to REST (return to DUPCO for writing exit message; Job complete).
FXAL	Go to GCOMM to read DCOM from disk and return. Update COMMA and DCOM for Fixed Area operation. Exit to PCOMM (to put DCOM to disk and exit).

STOREMOD (STMOD) Function

Chart: BH

- Stores a program or data on the disk overlaying another program or data of the same name.

This function accomplishes the replacement of a program or data stored on the disk by a new program or data without having to perform a DELETE function. It is assumed that the replacing program and the program to be replaced are in the same format. The word count of a new Core Image program is placed into the LET/FLET entry for that program. However, no change is made to LET/FLET when programs in Disk System format or Data files in Disk Data format are being stored.

The first sector of the User or Fixed Area which contains the program to be replaced is read into the output buffer. The first sector of Working Storage is read into the input buffer. The output pointer is made to point to the first word of the program to be replaced. The input pointer is made to point to the first word of the replacing program.

The input buffer is moved one word at a time to the output buffer until either the input buffer is empty or the output buffer is full.

When the output buffer is full, it is written onto the disk in the User or Fixed area. The next sector of the program to be replaced is read from the User or Fixed Area into the output buffer and the word for word replacement continues.

When the input buffer is emptied, the next sector of the replacing program is read into it from Working Storage and the word for word replacement continues.

This move and overlay sequence is repeated until all of the replacing program in Working Storage has been written onto the disk overlaying the program to be replaced.

This function then returns to DUPCO for initialization and the printing of the exit message.

Entry Point

SMODC+2	Exit to STMOV if not a Core Image program. Adjust the word count of the required entry in LET and write to disk.

Routines

STMOV	Save starting sector address and initialize output buffer. Initialize the move counters. Exit to STM4.
STM4	Initialize the input buffer and pointer. Exit to STM3.
STM3	Move 1 word from input to output buffer. Exit to STM1 if more room in output buffer. If output buffer full, then write buffer to disk. Reinitialize the output buffer and pointer. Exit to STM1.
STM1	Exit to STM2 if more words of this disk block still to be moved. Decrement count of disk blocks to be moved. Exit to STM2 if more disk blocks required to to be moved. Exit to REST (in DUPCO) if all words of output written back to disk. Write last output buffer to disk. Exit to REST (in DUPCO).
STM2	Exit to STM3 if there are more words in the input buffer. Exit to STM4.

DUMPLET Function

Chart: BI

- Restores the carriage before printing LET/FLET title.

- Prints title LET/FLET with a six-word or three-word header respectively.

- Prints one LET/FLET entry per line.
- Prints a five-word sector header for each sector after it is read into core.
- Prints the Location Equivalence Table (LET) and/or the Fixed Location Equivalence Table (FLET) on the principle print device.
- Returns control to DCTL, after last LET sector is printed, for printing an exit message.

When DUMPLET is first brought into core from disk, it determines whether LET, or both LET and FLET are to be printed. When printing LET, it first prints the title 'LET'. It then prints six words from COMMA as a header. These six words are: FPA, FPAD, COM6, COM12, COM4, COM10. After the first header is printed, a sector of LET is read from disk.

Within each sector of LET, there are five sector header words.

These five words are printed after each sector is read into core. These words include:

1. the sector ID number.
2. the last sector indicator.
3. the number of disk blocks used by the programs represented in this sector of LET.
4. the number of words that are available for entries in this sector.
5. the next LET or FLET sector address, or if this is the last LET or FLET sector, zero (0).

A LET entry can be either a three-word entry or a six-word entry. DUMPLET next determines whether it is pointing to a three- or six-word entry. If it is a three-word entry, it checks to see if it is a prime or non-prime entry. For a three-word prime entry, the disk block address is calculated by accumulating the previous disk block counts and adding them to COM6 (the base disk block address of the User Area). One LET entry will be printed per line on the print device. A printed prime entry will contain the program name, the disk block count of the program, and the starting disk block address of the program on disk. A non-prime LET entry will just have its entry name printed. For the six-word LET entries, a printed line will have the name of the entry, the disk block count, the disk block address, the core load address, the core execution address, and the actual disk block count of the program or Data file.

After the last LET sector is printed, control is given back to DUP Control, and a two-word exit message is printed. The first word of this message is the User Area base address. The second word is the accumulated disk block count of all the programs whose entries are in the LET table.

If it was determined from the DUMPLET control record that FLET was to be printed also at this time, DUP Control will return to DUMPLET and FLET will then be printed.

FLET, which is the map of the Fixed Area on disk, contains only six-word entries. These entries will be printed as described above for six-word LET entries.

The title 'FLET' is printed first, followed by a three-word header. The three words are obtained from COMMA words COM2, COM37, and COM5.

As in LET, there will also be a five-word sector header. This header will be printed after each new sector is brought into core.

At the completion of dumping FLET to the principal print device, control is given back to DUP Control, whereby the exit message is printed, similar to the one after dumping LET. Then a new control card is read and decoded.

Entry Point

DLETC+2	Branch to START.

Routines

START	This sets the LET/FLET header words switch (SCDSW) which is tested in the print I/O routine. Go to PRTRP to restore paper, and return. Set up the disk block address of the User Area base and initialize the exit message words. Check for FLET or LET printing. Exit to DFLT, if printing FLET; otherwise, go to PRTCR to space a line and return. Go to PRTEB to print the title 'LET' and return. Go to PRTCR to space a line and return. Set up for the five header words from COMMA. Go to PRTBI to print header words and return. Go to CLRBF-1 to clear LET buffer and return. Exit to SETUP.
SETUP	Set up the disk I/O buffer area LETAR with the word count and sector address of the LET sector. Exit to GTLET.
GTLET	Go to GET to get next sector of LET and return. Go to PRTCR to space one line and return. Check to see if this is a FLET sector; if so, exit to FLTHD. Exit to LTSKT.

LTSKT	Prepare sector header words for printing, and set the SCDSW switch. Go to PRTBI to print sector header words and return. Reset SCDSW. Go to CLRBF-1 to clear LET entry buffer and return. Go to PRTCR to space a line and return. Set up the number of words of the present sector to be printed. Exit to LSTSK if this is the last LET sector. Exit to SETLF if last LET sector, but there is a FLET to be printed. Exit to ENLUP if neither of these last LET sector conditions exists.
ENLUP	Exit to ENT6 if a six-word entry. Move the two-word entry name to the LET buffer LTBUF. Exit to MNONY if a non-prime entry. Calculate the disk block address of the current entry and place it in the LET buffer LTBUF along with the disk block count. Also update the second word of the exit message with the current entry's disk block count. Continue to BYPSS.
BYPSS	Exit to PRLET to print this entry, and return. Go to CLRBF-1 to clear LET entry buffer and return. Return to ENLUP if all entries in current sector have not been printed. Otherwise exit to TSTLT.
TSTLT	Exit to ENDMG if last LET sector. Exit to ENDMG if dump FLET indicator is on (RPSW). If neither condition above exists, restore XR3. Return to GTLET.
MNONY	Set the name only switch JSTNM.
CLRBF	A closed routine to clear the LET entry data buffer LTBUF. Return to calling routine.
SETFL	Set the print FLET indicator (RPSW). Exit to ENLUP.
LSTSK	Set the last sector indicator (LSTSW). Exit to ENLUP.
ENT6	Set the six-word entry indicator (SIXSW) and move the entry name to the LET buffer LTBUF. Disk block address of current LET entry is adjusted and moved to LET buffer LTBUF. Disk block address for next entry is calculated. Second word of exit message is updated. Exit to DATAN if this is a data entry name. Calculate CI program disk block count and move to LET buffer LTBUF. Exit to LST3.
DATAN	Exit to NODUM if this is not a 1DUMY entry. For a 1DUMY entry, move disk block count from third word of entry to LET buffer LTBUF. Exit to NODUM+1.
NODUM	Move disk block from sixth word of data entry to LET buffer LTBUF. Exit to LST3.
LST3	This moves the last remaining three words of a six-word entry to the LTBUF area. Reset SCDSW. Go to PRLET to print one LET entry and return. Reset SIXSW. Go to CLRBF-1 to clear LET entry buffer, and return. If all entries have been printed, exit to TSTL; if not, return to ENLUP.
FLTHD	Go to PRTEB to print title FLET and return. Go to PRTCR to space one line and return. Set up the FLET header words. Go to PRTBI to print FLET header words from COMMA and return. Go to PRTCR to space one line and return. Reset the FLET header switch FHDSW and return indirectly through FLTHD.
ENDMG	Go to PRTCR to space one line, and return. Exit to REST (return control to DUPCO).
DFLT	Set up the disk I/O buffer (word count and sector address) to get FLET sectors. Initialize the base disk block address of the Fixed Area, and reset the print FLET switch (RPSW). Set the FLET Header switch (FHDSW) and exit to GTLET.

Disk Write Address (DWADR) Function

Chart: BJ

- Rewrites sector addresses in Working Storage.
- Writes $D120_{16}$ into the first word of all sectors. Writes 2663_{16} into the second word of all sectors.
- Writes the next 238 words containing AXXX (where XXX = sector address in hexadecimal).
- Writes zeros in the remainder of the sector.
- Writes the above information on every sector from the start of Working Storage through Sector 1599 decimal ($63F_{16}$).

Entry Point

WADRC+2 Enter from PLUS2 routine of DUP Control. Insert the address of the last sector of the User Area into SAD. NOTE: SAD is used for the sector address. Exit to GET to read last sector of User Area and return. Thus the arm is positioned just one sector in front of Working Storage. Exit to ADSAD.

Routines

ADSAD Step SAD by one to obtain sector address of next sector to be written. Exit to NUCYL if this is a new cylinder. Exit to PUSAD.

PUSAD Format the sector; i.e., proper sector address, first two special words and the address ORed with A000 and filled out with zeros. Exit to the Disk I/O routine in the Skeleton Supervisor to cause the sector to be written and then return. Exit to ADSAD.

NUCYL If last sector of Working Storage has been written, exit to REST (return control to DUPCO). Use disk I/O to do a direct seek of plus 1 cylinder. Exit to PUSAD if not a defective track; otherwise, add seven to sector address in SAD. Exit to ADSAD.

DEFINE Function

Chart: BK

DEFINE consists of three basic segments which perform the following:

- Define a Fixed Area.
- Void the Assembler.
- Void FORTRAN.

To provide a Fixed Area or to increase its size, the Core Image Buffer, LET, and the User Area must be moved toward, and partly into, Working Storage. Working Storage will be reduced by the increased size of the Fixed Area. The corresponding addresses in COMMA and DCOM are updated. The elimination of FORTRAN or the Assembler will cause all programs, Core Image Buffer, and LET to be moved away from Working Storage and thus Working Storage will be expanded by the size of the eliminated program.

Both COMMA and DCOM must be updated accordingly. Once FORTRAN or the Assembler have been eliminated, neither can be restored without reloading the entire disk pack including all the programs in the User or Fixed Area that are still of value to the user.

Likewise, after a Fixed Area has been defined it cannot be reduced in size without a complete pack reload.

Entry Point

DEFC+2 Enter from DUP Control. Exit to VDASM if voiding Assembler. Exit to VDFOR if voiding FORTRAN. Exit to print error message if not sufficient number of cylinders requested (at least 2 if no previous Fixed Area, at least 1 if Fixed Area present). Exit to print error message if defining requested number of cylinders would exceed Working Storage. Set up addresses of first sector to be moved FROM and TO, using FPAD as a base address. Exit to MVDSK.

Routines

MVDSK Move sectors on disk left, until the FROM address is equal to the old CIB sector address (COM2). Exit to DKMVD.

DKMVD Update most of COMMA. Exit to PRVFA if there was a previous Fixed Area. Assign a FLET sector address (from old CIB). Exit to SHDR to set up the FLET sector header, and return. Exit to DUMMY to produce a dummy entry, and return. Write FLET sector to disk. Exit to FINSH.

FINSH Exit to UPDLT to update LET sectors. Exit to UPDCM to update DCOM. Exit to REST (return control to DUPCO for exit message and next DUP control record).

PRVFA Finds the address of the last sector of FLET. Reads the sector into core and locates the position of the last entry. If the last entry is a dummy entry, adds to it the number of disk blocks being defined (PDMY). If the last entry is not a dummy entry, checks if there is room for another (dummy) entry in this FLET sector. If yes, corrects the header and creates a dummy entry, using the closed subroutine DUMMY. If no, modifies the header and COMMA, writes the sector of FLET in core back onto the disk,

	builds a new sector header using closed subroutine SHDR, and creates a new dummy entry. Writes the sector onto the disk. Exits to FINISH.
VDASM	Print error message if Fixed Area present, or if Assembler is not present. Compute FROM, TO addresses of first sectors to be moved. Use closed subroutine MOVE to do all of the moving. Use closed subroutine UDCOM to update COMMA. Use closed subroutine UPDLT to update the LET sectors. Use closed subroutine UPDCM to update the DCOM sector. Exit to REST (return control to DUPCO).
VDFOR	Print error message if Fixed Area present or if FORTRAN is not present. Compute the FROM and TO of first sectors to be moved. Use MOVE to do the moving. Use UDCOM to update COMMA. Use UPDLT to update the LET sectors. Use UPDCM to update the DCOM sector. Exit to REST (return control to DUPCO).
DUMMY	A closed subroutine that inserts dummy entry into FLET. Set into words 1 and 2 the name code of 1DUMY. Set word 3 equal to the disk block count of the unused Fixed Area. Set words 4-6 to 0. Exit to calling routine.
UPDLT	A closed subroutine to update all the LET sectors. Get each LET sector. Modify word 5 of the sector header; that is, the sector overflow address. Put each LET sector back to disk in turn. Exit to calling routine.
UPDCM	A closed subroutine to update DCOM. Get DCOM from disk. Insert into DCOM the corresponding values from COM2, COM5, COM6, COM12, COM35, COM36, COM37, COM52, COM54, FILE, FPA, FPAD, ASAD, and FSAD. Put DCOM back to disk. Exit to calling routine.
UDCOM	A closed subroutine to update COMMA. Alter COM2, COM6, COM12, COM35, COM36, FILE, FPA, and FPAD by the required amount. Exit to calling routine.
MOVE	A closed subroutine to move specified sectors from sector whose address is contained in FAD2 to sector whose address is contained in TAD2 and continue moving until last sector to be moved has been moved. Exit to calling routine.

DUP I/O

Because of the variety of I/O devices and formats required, specialized I/O routines are used in DUP. These routines consist of the basic Read and Write IOCS, the normal device code conversion routines, and the special formatting routines.

At system load time, the System Loader/Editor selects the required I/O routines based on the system configuration records. These I/O routines are loaded to the disk in specified sectors. The ILS routines required by these I/O routines are grouped within DUPCO to ensure they are in core during a DUP operation. In addition, since it is desirable to permit paper tape output while the principal I/O is cards, the paper tape I/O routine (PTX) is loaded in the auxiliary IOCS cylinder regardless of which I/O routine is loaded in the principal IOCS cylinder.

Thus, at DUP execution time, any DUP function requiring I/O specifies a word count and transfers to an I/O entry point determined by the format to which the data or program is to be converted. That entry point in any DUP I/O routine accomplishes the required conversion and the formatting of the data or program as dictated by the entry point.

Table 1 shows the I/O code conversions accomplished within the DUP I/O routines. See the publication IBM 1130 Monitor System Reference Manual (Form C26-3750) for detailed descriptions of the various formats handled by the DUP functions.

DUP I/O ROUTINES

DISK0 Routine

- Performs all the disk I/O functions for DUP.

This routine is described in the publication IBM 1130 Subroutine Library (Form C26-5929).

CARDX Routine

- Performs the card I/O functions for DUP including the required conversions.

Entry Points

GETCD	A closed routine to read a card in IBM card code. Store a count of 80 into core I/O area, read one card, using card I/O routine, into CRBUF. Convert the 80 characters into 40 packed EBCDIC words in CRBUF-40 using the Speed Routine.

Table 1. I/O Code Conversion Table

Disk Representation	Input or Output		Output	
	Card	Paper Tape	1132 Printer	Console Printer
1 word (16 bits) packed EBCDIC	2 columns (2 characters) IBM card code	4 frames (2 characters) PTTC/8 code	2 alphameric characters	2 alphameric characters (via Rotate/Tilt code)
1 word (16 bits) binary	1-1/3 columns binary	2 frames binary	4 alphameric characters-hex number	4 alphameric characters-hex number (via Rotate/Tilt code)

CDGET — Routine to read a binary coded card. Read 72 columns into DF-81 through DF-10. Pack the 72 words into 54 binary words at DF-81. If the card is a data card (DATSW ≠ 0), skip the check sum routine. Check sum routine: If check sum word in card is zero, don't do check sum. If check sums don't compare, exit to SEXIT (error exit in DUPCO). Exit to calling routine.

CDSEL — A closed subroutine to pass non-DUP, non-SUP cards. Stacker select the last card read. Exit to calling routine.

CDPCH — A closed subroutine to punch binary cards. The card ID is in core in name code. Convert ID to IBM card code and store in card area. If card is a data card, skip check sum; otherwise calculate the check sum and store it in word 2 of the card area. Convert sequence number from binary to IBM card code and store in card area. Expand binary to 12-bit words. Punch 80 columns. Update sequence number for next card. Exit to calling routine.

Buffers

DF-82 — Binary card input.
CRBUF — IBM card code input.
CRBUF-40 — Packed EBCDIC card buffer.
PCH1 — Binary card output (12 bit).
RD1 — Binary card output (16 bit).

TYPX Routine

- Performs the output functions to the Console Printer for DUP including conversions.

Entry Points

PRLET — A closed subroutine to print LET. Set up for TRUEB and PRTBI. Convert LET entry from name code to fine EBCDIC characters using TRUEB. If entry contains a name only, type name using PRTEB and exit to calling routine. If entry is more than just a name, set up for EBPRT conversion. Convert entry from EBCDIC to printer code using BINHX and HOLPR. Set up to print binary. Print entry using PRTBI. Exit to calling routine.

PRTBI — A closed subroutine to print binary. Set up and convert data using BINHX. If the correct number of words has been printed, go to PTLN and then exit to calling routine. PTLN converts data to printer code using HOLPR and prints the line. If the end of the sector being printed is reached, get next sector. If the end of a four-word group is reached, put in an extra space and check for a full line. If line is full, go to PTLN,then back to start of PRTBI; otherwise, go to set up and convert with BINHX.

PRTCR — Set up for carriage return. Go to WRTY, then exit to page restore routine.

PRTRP	Go to PRTCR 10 times, then exit.
PRTEB	A closed subroutine to print packed EBCDIC. Go to PRTCR to return carriage. Set up and convert data using EBPRT. Print a line using WRTY routine, then exit to calling routine.

Buffers

RD2+1	Packed EBCDIC in TRUEB; IBM card code in BINHX.
RD3	Output buffer for Console Printer code when printing LET and binary.
PRNTB+40	Buffer for Console Printer code when printing EBCDIC.

VIPX Routine

- Performs the output functions to the 1132 Printer for DUP including conversions.

Entry Points

VPSP	A routine to set up a carriage space. Check for 1132 ready and not busy. XIO to start carriage space. Exit to calling routine.
VPSK	A closed subroutine to set up a carriage restore. Check for 1132 ready and not busy. XIO to start carriage skip. Exit.
VPET	Routine to print LET. Set up TRUEB and convert LET entry from name code to EBCDIC. If entry contains a name only, print the name using VPEB and exit. If entry contains more than just a name, set up and convert rest of line. Print entry using VPBI. Exit to calling routine.
VPBI	Routine to print binary. Set up and convert data using BINHX. If the correct number of words has been printed, go to PTLN, then exit (PTLN prints the line). At end of a sector, space two extra lines. If a four-word group is complete, put in an extra space. Check for a full line. If full, go to PTLN then back to BINHX.
VPEB	Routine to print packed EBCDIC. Expand packed EBCDIC to EBCDIC. Print lines using VIP printer special. Exit to calling routine.

Buffers

RD3+16	Output buffer for unpacked EBCDIC.
RD3	Output buffer for printing LET and binary.

PTX Routine

- Performs all paper tape I/O functions for DUP including conversions.

Entry Points

GETPT	A closed routine to read PTTC/8 record. Sets word count for paper tape read to 80. Reads one record, with check, using PAPT1. Sets character count for paper tape conversion to 160. Converts from PTTC/8 to packed EBCDIC using PAPEB. Conversion will stop after DD (new line) code. Return to calling routine.
PTGET	A closed routine to read a binary record. Reads one word, with check, which brings in hexadecimal 00 word count (WC). Reads WC words without check. If record is not a data record (DATSW = 0), do a check sum on record, then compare the check sums.
PTSEL	Dummy entry point corresponding to CDSEL in CARDX routine.
PTPCH	Routine to punch binary records. If record is a data record, set the word count to 54 and skip check sum routine. If record is not a data record, determine the word count (WC) by closing off blanks from the right; calculate a check sum. Punch hexadecimal 7F followed by the record. Exit to calling routine.

Buffers

CRBUF	PTTC/8 control record input.
CRBUF-40	EBCDIC area after conversion.
DF-82	Binary tape record input.
RD1-2	Binary tape record output.

Error Message Routine (PERRC/TERRC)

- Prints the appropriate error message.
- Terminates the DUP function.

To perform the printing of the error messages, the routine PERRC is present in those systems having the 1132 Printer. In all other systems the routine TERRC is present. The routines function identically. The System Loader selects the required routine at system load time.

Entry Point

There is only one entry point, PERRC+2/TERRC+3, which is the first word of the first sector of PERRC/TERRC. Control is passed to the error message routine after a DUP function has requested an error message by way of a long BSI instruction to SEXIT. The BSI is followed by a DC containing the error parameter.

PERRC/TERRC

Using the error parameter, this routine calculates (1) the word count of the message which is placed into XR1, (2) the error message number which is placed into XR2, and (3) the return address. The print routine contained within PERRC/TERRC prints the requested error message and returns to the address specified.

Exits

If the message printed is "D03 Invalid Header Length", then the calling DUP function was DUMP or STORE. In this case, control is not returned to DUMP or STORE. Instead, records are read until either (1) a Monitor control record is encountered and a call to the Monitor is effected after setting the non-read switch or (2) a DUP control record is encountered, in which case an exit is made to REST+1 in DUPCO which initializes the DUP function specified in the control record read.

If the message printed is "D72 Load Blank Cards", control is returned to the calling DUP function, DUMP, which then continues after more blank cards have been loaded.

In all other cases, control is returned to REST+1 in DUPCO, the exit message is thus inhibited, and DUPCO initializes for the next DUP function.

See the publication IBM 1130 Monitor System Reference Manual (Form C26-3750) for a list of the error messages, their meanings, and the routines from which they are called.

This section contains descriptions of the internal structure and operation of the 1130 Monitor System Assembler.

The 1130 Monitor System Assembler is designed to translate the statements of a source program written in 1130 Assembler Language into a format which may be dumped and/or stored by DUP or executed directly from Working Storage.

Basically, the functions of the Assembler are:

1. Convert the mnemonic to machine language (except for Assembler control records).
2. Assign addresses to statement labels.
3. Insert the format and index register bits into the instruction if applicable.
4. Convert the instruction operands to addresses or data.

PROGRAM OPERATION

Phase 0 of the Assembler Program is read into core storage by the Supervisor whenever the Supervisor encounters an ASM Monitor control record. Control is then transferred to the Assembler Program.

The source program is read and processed (one statement at a time) twice during each assembly. The source statements are normally read into core storage from the principal I/O device, processed, and stored on disk during the first Pass (Pass 1).

During the second Pass (Pass 2) the statements are read from the disk for processing. When the source program is read in only once, the assembly is said to be in one pass mode. In some cases, the source program is read through the principal I/O device on Pass 1 and Pass 2. The assembly is then said to be in two pass mode.

Pass 1

The Supervisor reads the first sector (Phase 0) of the Assembler Program into core storage and transfers control to it. Phase 0 then reads in the principal I/O and principal printer subroutines and a level 4 ILS, sets up the Interrupt Transfer Vector (ITV), reads in the non-overlay section (Phase 9) of the Assembler and then overlays a part of itself with the Control Record Phase (Phase 1). The control records are read, analyzed, and listed. Switches are set for the various options specified by the control records. When the first non-control record statement is encountered, Phase 1A is read in overlaying Phase 1. If the principal input device is the paper tape reader, Phase 1A moves the source record previously read 20 character positions to the right in the input buffer. It then initializes the boundary conditions for the Symbol Table. When this is completed, Phase 2 is read in, and statement processing begins.

As the statements are processed, a table is built which contains the symbols found in the Label Fields. This table is called the Symbol Table. Each valid label used in the program along with the address assigned to the label is entered into the Symbol Table. The format, contents, and use of the Symbol Table are described in detail later in this section. The building of the Symbol Table is the major objective of the Assembler during Pass 1.

Pass 2

At the end of Pass 1, the Assembler initializes itself for Pass 2. Phase 1 is read in; S1A +2 is the entry point for Pass 2. If the Assembler is operating in one pass mode, Pass 2 begins automatically, using the source program stored on the disk during the first pass. If the Assembler is in two pass mode, the source program must be reloaded to begin Pass 2. During Pass 2, object output in the form of hexadecimal digits and/or error codes is generated. The object output is stored on the disk in Disk System Format and an on-line listing may be obtained. If a list deck or paper tape object program is desired, the Assembly must be made in two pass mode.

When the input is from cards and a list deck is specified, the output is punched into the first 19 columns of each source card. The object output consists only of error codes if the List Deck E option is specified.

When the input is from paper tape, the entire source statement is punched into the new tape in addition to the object data. (This simulates the operation of the 1442 Card Reader Punch.)

I/O Data Flow

The input to the Assembler is from cards or paper tape. The output depends upon the mode of assembly. (See Figure 10.)

One Pass Mode: In Pass 1, source program statements are read into storage and stored on the disk. The data in the I/O area is packed 2 EBCDIC characters per word, beginning with the first position of the label field if the label field is not blank, or with the first position of the operation field if the label field is blank. A prefix word which indicates the word count and the starting point of the record (Col. 21 or 27) is added before placing the statement on disk.

During Pass 2, the statements are read from disk, unpacked, and processed. The output object data is stored on disk in Disk System Format. In addition, a printed listing of the object program and/or the Symbol Table may be obtained by specifying these options with control records.

Two Pass Mode: The card or paper tape source program must be read into core storage on Pass 1 and Pass 2. During Pass 2, object data are stored on the disk. A list deck, not compressed, or tape and/or a listing will be generated if specified by an Assembler control record. A listing of the Symbol Table and/or Symbol Table deck may also be obtained if specified by Assembler control records.

RELOCATABILITY

The Assembler Program will assemble programs in either absolute or relocatable mode. All subroutines must be assembled in relocatable mode.

Absolute Mode

An absolute program is one that is loaded into a predetermined area of core storage for execution. The core locations used at execution time are the same as those specified in the source program.

In an absolute assembly, all values assigned to labels are absolute and all operands are absolute.

Relocatable Mode

A relocatable program is one that can be loaded, by the Loader, into (and executed from) an area of core independent of the addresses assigned to instructions and constants during assembly.

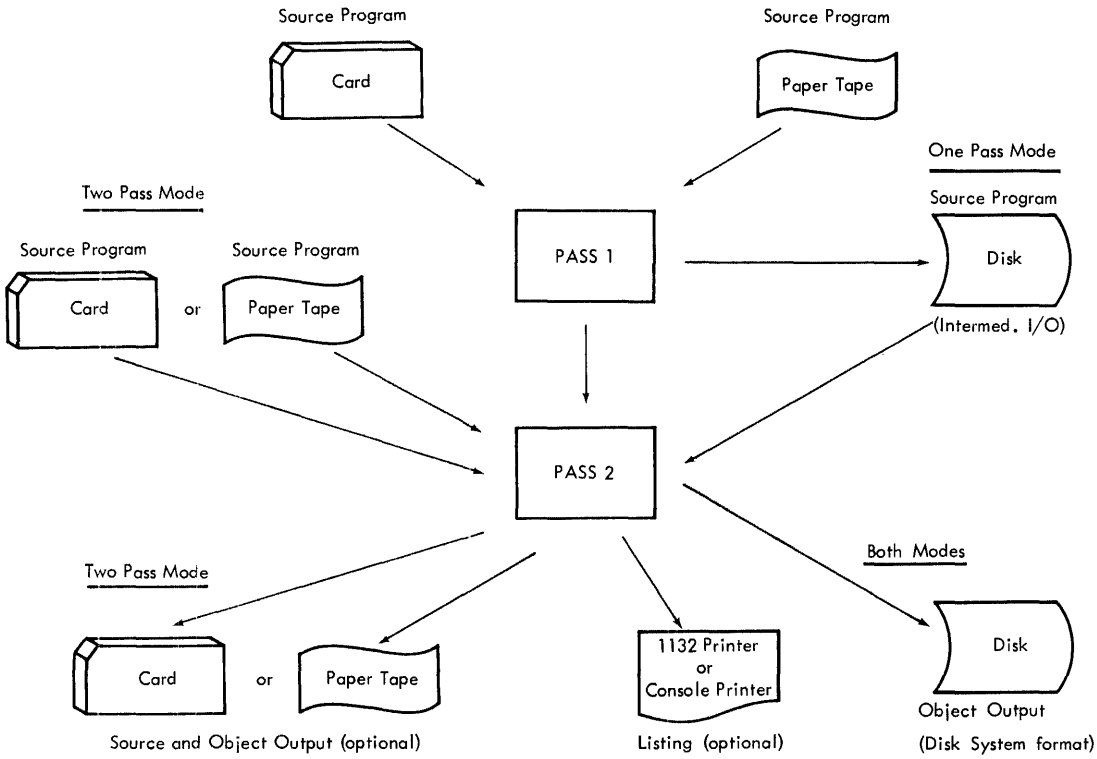


Figure 10. Assembler I/O Flow

Labels

When labels are entered into the Symbol Table and the assigned value is relocatable, a bit signifying that it is relocatable is also entered. All labels except EQU statement labels are relocatable. EQU statement labels are absolute if the operand is absolute and relocatable if the operand is relocatable.

Operands

Operands that are assembled to become the second word of a two-word instruction or an address constant may be assembled relocatable. These words will be modified by the Loader at load time.

The Loader will modify the second word of relocatable long instructions and the value of relocatable constants by the difference between the address assigned to an instruction and the address that the instruction will actually occupy. Operands that must be relocatable in a relocatable program are:

1. The operand of the second word of a two-word instruction when it represents the address of an instruction, a constant, or data that is contained within the program.
2. The operand of a short instruction if the displacement is to be added to the IAR contents to yield the effective address at execution time. This includes all instructions assembled with blank format and tag fields except LDX, LDS, and any shift instruction.
3. The operand of a DC statement when it represents the address of an instruction, a constant, or data contained with the program; for example, a constant that represents the address used by an indirect addressing instruction.
4. The operand of an ORG, ENT, or END statement.
5. The operand of an EQU statement if it is the label of a relocatable statement.

All other operands must be absolute.

Determining Relocatability: The Assembler adds one to a counter for each relocatable element that is added in the operand and subtracts one for each element that is subtracted. When two or more elements are multiplied (only one of which is relocatable) the value of the absolute element (or product of the absolute elements) is added to (or subtracted from) the counter. When all operand elements have been processed, the counter must be equal to +1 (relocatable) or 0 (absolute). Any other value will result in a relocation error. Relocatable elements in operands are:

1. A symbol that is defined by means of the Location Assignment Counter (see Tables and Buffers).
2. The asterisk symbol when used as an element.

Absolute elements in operands are:

1. Decimal and hexadecimal integers.
2. Character values.
3. Symbols which have been equated to an expression which has an absolute relocation property.

NOTES

1. A symbol that has been equated (by means of the EQU Assembler instruction) to an expression assumes the same relocation property as that expression.
2. The difference of two relocatable quantities is an absolute quantity.
3. A relocatable quantity plus or minus an absolute quantity is a relocatable quantity.
4. The sum or difference of any number of absolute quantities is an absolute quantity.
5. Relocatable elements cannot be multiplied by relocatable elements.
6. In an absolute assembly, all symbols and the asterisk value are defined as absolute; therefore, no relocation errors are possible.

STORAGE LAYOUT

Figure 11 illustrates the layout of core storage during an assembly. The addresses listed are approximate and should not be construed as the final address assignments. For the actual addresses, refer to the program listing using the symbolic labels.

Address	Segment
0000	Skeleton Supervisor: including ITV, COMMA, Disk ILS, DISK0
025C	DBUF-Card or P.T. I/O; object output buffer (one pass mode)
03F4	BUFI-Intermediate I/O buffer or object output buffer (two pass mode)
0542	ILS04, DISK1, multi-sector subroutine, FLIPR
05B2	Overlay Area 1. Phase 0 Assembler Loader 2. Phase 1 Control records 3. Phase 1A 4. Phase 2 First card group 5. Phase 5, or 6, or 7, or 8, or 12 6. Phase 12A 7. Phase 3 Symbol Table Output 8. Phase 4 Last phase
0774 07B7	BGASM 0P0K Non-overlaid Mainline (Phase 9)
07D6 07D8 07DA 07DC 07DE 07E0 07E1	GETS6 GETS7 GETS8 GETS2 GETS5 GTS12 End of Phase 3
07F7 0925 09E0 0A67 0A6F 0AB3 0B42 0B92 0BE9	SCAN BEG0P SAREA (80 word input buffer) B4HEX ERFLG LDLBL LABCK DF0UT STSCH
0C81	STADD Overlayed by DTHDR and WRDF0 (Phase 10)
0D0B	INT1 Overlayed by INT2 (Phase 11)
0DDB	Optional Print Routine: Console Printer or 1132 Printer
0EB2*	One sector buffer for Symbol Table overflow
End of Core	In-Core Symbol Table

*Other possible lower limits for Symbol Table
1. 0DD8: No listing, one pass mode
2. 0D74: No listing, two pass mode

Figure 11. Assembler Storage Layout

OUTPUT FORMAT AND ERROR CODES

Table 2 lists the contents of character positions 1-16 (each character position corresponds to one word in the I/O buffer) of the I/O buffer. The contents are inserted during Pass 2 in hexadecimal format. Just prior to the optional print and/or punch operation, the buffer contents are converted to the proper output format for the output devices.

Character positions 18 and 19 of the I/O buffer are used for error codes. The code for the first error detected in an erroneous statement is inserted into position 18. The code for any remaining errors is inserted into position 19. Note that if three or more errors are detected within one statement, only the first (in position 18) and the last (in position 19) are indicated. The error codes are listed in Table 3.

Table 2. Format of the I/O Buffer

Statement Type	I/O Buffer Position*	I/O Buffer Value in Hexadecimal
ABS	1-16	Blank.
BSS, BES	1-4	Location assigned to the label, if any.
	9-12	Number of reserved words (operand value).
CALL	1-4	Location assigned to the branch instruction built by the Loader.
	6-7	30 (special relocation code for a CALL).
	9-16	Subroutine name in packed EBCDIC form.
DC	1-4	Location assigned to the constant.
	6	Relocation code for the constant (0 = absolute, 1 = relocatable).
	9-12	Value of the constant.
DEC	1-4	Location of the left-most word of the constant (always at an even location).
	6-7	0 0 (both words are absolute).
DSA	1-4	Same as CALL.
	6-7	31 (special relocation code for a DSA).
	9-16	Same as CALL.
EBC	1-4	Location of the left-most word of the operand.
	9-12	Number of core positions reserved for the operand.
END	1-4	Next available even location after this program.
	9-12	Starting address if mainline program.
ENT	1-4	Location of the entry point.
	9-12	Name of relative entry point.
EPR	1-16	Blank.
EQU	1-4	16-bit operand value.
EXIT	1-4	Location assigned to the label.
	6	0.
	9-12	60XX (XX is the address of the EXIT entry point to Skeleton Supervisor; i.e., 6038).
HDNG	1-16	Blank.
All Imperative Instructions	1-4	Location of the left-most instruction word.
	6	Relocation code for word 1 (always zero).
	7	Relocation code for word 2 (blank for a short instruction; 0 = absolute, 1 = relocatable).
	9-12	Machine-language OP code, F and T, and Displacement.
	13-16	Word 2 of long instructions.
ISS	1-16	Same as ENT.
LIBF	1-4	Same as CALL.
	6-7	20 (special relocation code for an LIBF).
	8-16	Same as CALL.
LIBR	1-16	Blank.
LINK	1-4	Location assigned to label, if any.
	6-7	00.
	9-13	Same as CALL.
ORG	1-4	Location assigned to the label, if any.
SPR	1-16	Blank.
XFLC	1-6	Location of the left-most mantissa word.
	6-7	Value of the exponent.
	9-16	Value of the mantissa.

*Buffer positions correspond directly with card columns in the List Deck. Unlisted positions are always blank.

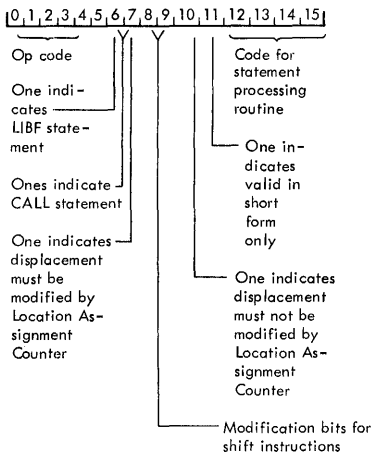
Table 3. Assembler Error Codes

Error Code	Error Description	Error Procedure
A	Address Error An attempt has been made to specify a displacement, directly or indirectly, outside the range +127 to -128	The displacement is set to zero.
C	Condition Code Error A character other than +, -, Z, E, C, or O (Alpha) has been encountered in a condition operand.	The displacement is set to zero.
F	Format Error A character other than a blank, X, L, or I has been used in the format field or an L or I format has been specified for an OP code valid in short form only.	The instruction is processed as a long instruction if the instruction is valid in the long form. Otherwise, it is processed as though an X format code had been specified.
L	Label Error An invalid symbol has been used as a label.	The label is ignored.
M	Multiply-defined Symbol Error More than one statement has the same symbol in the label field.	The first occurrence of the symbol defines the value in the Symbol Table. Subsequent occurrences of the same symbol are ignored. Note: this error will appear in statement referencing the multiply-defined symbol.
O	OP Code Error The mnemonic OP code is not in the OP code table. Header type statements are incorrectly positioned in the source program.	The Location Assignment Counter is incremented by two. The statement is treated as comments (except for the error code).
R	Relocation Error The operand is neither absolute nor relocatable, or two relocatable operand elements are multiplied by each other.	The effective operand value is set to zero.
	The operand is absolute in a relocatable assembly when the displacement had to be modified or the operand is relocatable when the displacement did not require modification.	The displacement is set to zero.
	The operand of an ENT or ISS statement is absolute.	The effective operand value is set to zero.
	The operand of an ORG statement is absolute in a relocatable assembly.	The operand value is ignored.
	The operand of a BSS or BES statement is relocatable.	The effective operand value is set to zero.
S	Syntax Error Illegal syntax in the operand field (e.g. invalid symbols, adjacent operators, illegal characters in an integer, no 'before character values).	The affected operand is given a value of absolute zero.
	Mainline program entry point not specified in an END statement of a mainline program.	Positions 9-12 of the I/O buffer are left blank.
	Incorrect syntax in a DEC or XFLC operand (e.g. Illegal character, loss of high order bits, exponent overflow).	Constant set to 0 or 0.0.
T	Tag Error The Tag field in an instruction contains a character other than a blank, 0,1,2, or 3.	The Tag field is set to zero and the statement is processed as though the Tag field were zero.
U	Undefined Symbol Error An Undefined symbol is found in an operand.	The affected expression is given the value of absolute zero.

TABLES AND BUFFERS

Operation Code Table (BEGOP)

The operation code table contains three words for each mnemonic entry. The first two words of each entry contain the four characters of the op code (packed 2 EBCDIC characters per word). The third word contains the binary machine-language op code and information used by the routine that processes the particular op code. The format of the third word is as follows.



The statement-processing routine code (bits 12-15) is used to branch to a branch table at the beginning of each instruction-processing overlay. If the overlay required for this instruction is in core, the branch table will branch to the specific routine. Otherwise the branch is to the mainline to set-up the read of the required overlay. See Figure 11.

Instruction Buffer (INSBF)

The instruction buffer is a one-word work area used when instructions are being formed. It contains only the first word of long instructions. Initially, it is loaded with the op code (bits 0-4) and the modification bits (bits 8-9) for shift instructions (derived from the third word of the op code table). As the instruction is being processed,the format, tag, indirect addressing, and displacement bits are inserted.

At the end of the DISP routine, the contents of the instruction buffer are saved in the one-sector DSF output buffer and then are converted to four EBCDIC characters (four bits per character) and stored in positions 9-12 of the I/O buffer.

Location Assignment Counter

The Location Assignment Counter (named address counter in the listings; symbolic:ADCOW) is a one-word counter used to assign sequential storage addresses to the program statements. It always contains the next available address.

The counter is initially set to zero and is set differently or incremented according to the statement type as shown in Table 4.

Secondary Location Assignment Counter

The secondary Location Assignment Counter (ADCW2) is used to detect breaks in sequence in Disk System Format output. It is incremented by one in the DFOUT subroutine for every data word entered in the output buffer (except the first word of an LIBF entry point name).

Table 4. Location Assignment Counter

Statement Type	Effect
ABS	Set to the lowest loadable core address. (Address obtained from the symbolic location LDRND.)
BSS, BES	Incremented by the value of the operand. (If E-format and the counter is odd, increment one more.)
DC, LIBF	Incremented by one.
DEC	Incremented by two. (If the Location Assignment Counter is odd, increment one more.)
DSA, XFLC	Incremented by three.
EBC	Incremented by one-half the number of operand characters. (An odd character count is incremented by one before the count is halved.)
END	Incremented by one if the Location Assignment Counter is odd.
LINK	Incremented by four.
ORG	Set to the value of the operand.
EXIT	Incremented by one.
Invalid OP Code	Incremented by two.
Machine Instruction Statements	
Short	Incremented by one.
Long or Indirect	Incremented by two.

Symbol Table

The Symbol Table is a table containing the source statement labels and their assigned values. There are also bit positions to indicate that the label is relocatable or multiply-defined.

All symbols defined in the program are entered in the Symbol Table. Symbols that appear in the label field of Assembler instructions which do not use labels (for example, ABS, END, ENT) are not entered.

The Symbol Table begins at the high address end of core and extends toward one of three lower limits (see Figure 11):

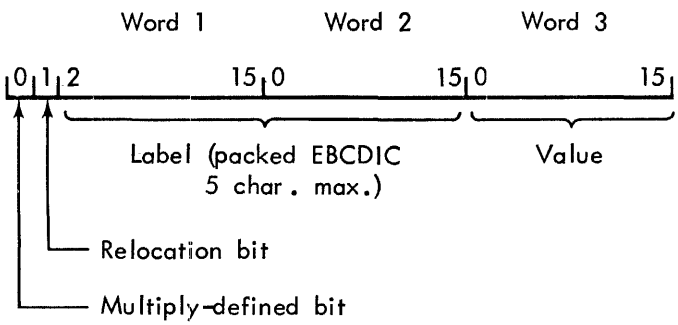
1. The end of the print routine when LIST option selected. The Console Printer and 1132 Printer routines are not the same length, but the lower limit will be adjusted for the print routine loaded at assembly time.
2. The end of Phase 11 when LIST option is not selected. The Symbol Table is allowed to overlay the print routine.
3. The end of Phase 10 when LIST option is not selected and there is no intermediate I/O (two pass mode).

Symbols are added to the Symbol Table in alphanumeric order with higher values (Z9999) toward the lower address end. If the lower limit is reached, a one-sector buffer is written on the disk to allow more symbols to be added. This buffer is at the low address end of the table. Each overflow sector is therefore ordered; however, there is no ordering between overflow sectors. There may be up to 32 overflow sectors.

Symbol Table Size (Approximate)

Size of Core (Words)	LIST	No LIST 1 PASS	No LIST 2 PASS	Max., with Maximum Overflow
4096	111	184	217	3609
8192	1476	1549	1582	4974

Each entry in the Symbol Table requires three words. The format of a Symbol Table entry is:



During Pass 1, labels are entered into the Symbol Table by the Symbol Table Add Routine (STADD). As each label is processed, the partially built Symbol Table is searched to determine if the label has been previously defined. If it has been previously defined, the multiply-defined bit (of the first entry) is set, and the label is not entered.

If the label has not been previously defined, it is entered along with the current value of the label value buffer.

If the program is being assembled in relocatable mode (no ABS statement), the relocation bit is set for each relocatable label. (See Relocatability.)

The Symbol Table is used during Pass 2 when evaluating an operand containing symbols. The symbol in the operand field is given the value of the symbol in the Symbol Table. If the multiply-defined bit was set, an M (multiply-defined error code) is entered in the I/O buffer.

Card Code Input Conversion Table

Conversion from IBM Card Code to EBCDIC is done by table lookup. The conversion table contains two EBCDIC characters per word with all 256 characters represented. The leftmost eight bits of each word represent the card code characters that can be formed with 12 through 8 punches; the right-most eight bits of each table word represent the card code characters that can be formed with 12 through 9 punches. Thus, if the input character contains a 9 punch, the right half of a conversion table word will be used. Conversely, if the input character does not contain a 9 punch, the left half of a conversion table word will be used. (In searching for the proper word, the 12-8 punches are used to decide which word; the 9 punch is used to determine which half of the chosen word.) (See TLU subroutine description.)

Paper Tape Input Conversion Table

The conversion of paper tape input to EBCDIC is performed by table lookup. The input character is used as the argument to perform a table search for the equivalent EBCDIC character.

The conversion table contains only valid PTTC/8 characters (73 total) and the EBCDIC equivalent. Bits 0-7 of each table word contain the EBCDIC character code and bits 8-15 contain the equivalent PTTC/8 character code.

PHASE DESCRIPTIONS

The generalized logic flow of the Assembler is shown in Chart BL. The labels beside the chart symbols correspond to the labels used in the program listings.

The following phase descriptions are divided into the routines represented in Chart BL. Subroutines are described with the phase in which they reside.

When the Supervisor encounters a ASM Monitor control record, Phase 0 of the Assembler is read into core storage and control is transferred to it. Phase 0 sets up the ITV for the principal printer and I/O device and reads in the ISS subroutines for these devices. It also initializes the Symbol Table limits, initializes the heading sector on CIB, makes sure there are 33 sectors of Working Storage available, reads in Phase 9, and reads in Phase 1, which overlays part of Phase 0.

Phase 0 also contains a level 4 ILS subroutine, a routine that is used with the DISK0 subroutine to enable it accomodate word counts exceeding 320, and a flipper routine that uses the disk routine to read in overlay phases. These routines remain in core throughout the assembly.

Phase 1 reads and processes control records, sets switches for each control record specified, and executes a branch to the print routine to print each control record. When the first non-control record is encountered, Phase 1A is read in, overlaying Phase 1. Phase 1A modifies the I/O area if paper tape input is used, and during Pass 1 determines the lower limit of the Symbol Table on a basis of the options specified by the control records. Phase 2 is then read in, overlaying Phase 1A. Phase 2 processes ABS, ENT, ISS, ILS, LIBR, EPR, SPR, and HDNG statements. When any other type of statement is encountered, Phase 6 is read in, and a branch to Phase 9 is executed.

Phase 3 is used to save the Symbol Table (optional) and to print and/or punch the Symbol Table (optional). It is read in from the disk at the end of the Pass 2 processing of the END statement. Upon conclusion of Phase 3, FLIPR is set up to read Phase 4 from the disk.

Phase 4 is used to print the closing message of the number of errors in the assembly and to move the object program output back 4 cylinders (to the beginning of Working Storage). This phase is read from the disk at the end of Phase 3 and returns to the Monitor entry point in the Skeleton Supervisor when completed.

Phase 5 processes ORG, EQU, HDNG, BSS, and BES statements. It also contains the subroutine used to print the heading at the top of each new page of the listing. The phase is always in core as a result of one of the above statement types, except when brought in as a result of a Channel 12 condition on the 1132 Printer.

Phase 6 processes all imperative instructions (hardware mnemonic op codes) and the DC statement. The phase is always in core as a result of an imperative instruction or a DC statement.

Phase 7 processes DEC and XFLC statements. The phase is always in core as a result of encountering a DEC or XFLC constant.

Phase 8 processes CALL, LIBF, DSA, LINK, EXIT, and EBC statements. The phase is always in core if one of the above statements is encountered.

Phase 9 checks the mnemonic op code for all statements except ABS, ENT, ISS, ILS, LIBR, EPR, and SPR statements, and executes a branch to the branch table contained in each statement processing phase. If the required phase is in core, the statement is processed. If the required phase is not in core, it is read in, and the statement is processed. Phase 9 also contains subroutines that are common to all phases of the Assembler.

Phase 10 consists of two subroutines used during Pass 2 to control the generation of the Disk System Format output. This phase is an overlay which is read over the Symbol Table Add portion of Phase 9 during Pass 1 of the END statement processing.

Phase 11 contains a subroutine that is used to read the source statements from the disk during Pass 2 if the assembly is in one pass mode. It is read in when the END statement is processed during Pass 1 and overlays the subroutine used to save the source statements on the disk during Pass 1.

Phase 12 is brought into core when the END statement is encountered. This Phase is used in Pass 1 to build the program header record, read in Phase 10 to replace the STADD section of Phase 9, and read in Phase 11 to replace a section of Phase 9 (one pass mode only), and then overlay with Phase 1 after the first record for Pass 2 is in the I/O buffer. In Pass 2, Phase 12 reads in Phase 12A to process the END statement, finishes the DSF output, and then Phase 12A is overlaid.

Phase 0 (Initialization Phase)

Chart: BM, BN

- Contains the non-overlay routines, ILS04, DISK1, and FLIPR.
- Reads in non-overlay phase (Phase 9).
- Reads in ISS subroutines for the principal printer and I/O device.

- Initializes: ITV, Symbol Table Limits, and the listing page heading on the first sector of the CIB.

- Checks Working Storage available on disk.

Non-Overlay Section of Phase 0

ILS04: Interrupt level subroutine for level 4. This subroutine senses the interrupt level status work (ILSW) for level 4 to determine which device on this level is responding. When this is determined, a BSI instruction is executed to the interrupt service entry point in the ISS subroutine for the device. When the ISS subroutine returns to ILS04, the interrupt is reset and the routine returns control to the mainline program.

DISK1: Multiple sector read/write subroutine. This routine is used with DISK0 to read and write more than 320 words at a time. The two words preceding the I/O area are saved and 320 is subtracted from the word count. If the word count is greater than 320, 320 is used as word count and a read or write of one sector is performed. If the word count is less than 320, the specified amount is used and the read or write operation is executed. When the disk operation is completed, the two previously saved words are restored. If the last word count was not less than 320, the subroutine sector address is incremented by 1, the next two words are saved, and a read or write function is executed. This process is repeated until the word count indicates that all data has been read or written.

DSKER: Disk error exit.

TV3 (Pseudo-transfer Vector Entry): Replaces the need for an LIBF statement. Branches to the standard entrance in the Print subroutine.

FLIPR (Overlay Flipper): Uses DISK1 to read in overlay phases. The sector address and word count are set up before entering FLIPR.

BRBCK (Branch Back): A long branch that is modified to branch to the correct location in the branch table of an overlay phase.

OVRLY (Overlay): Contains the word count and sector address of the overlay phase now in core or to be read into core.

Overlay Section of Phase 0

STRTO: Start of Phase 0 execution. Also corresponds to the loading address of all instruction-processing phases.

1. Save the settings of non-XEQ and non-DUP switches and set them on temporarily.
2. Set up interrupt level addresses for levels 0, 2, and 4 (words 8, 9, and 12 of the ITV).
3. Use the DISK1 routine to read in Phase 9. Use the DISK1 routine to read in the ISS subroutine for the principal printer and the principal I/O device.
5. Set the HIEND and LOEND of the Symbol Table.
6. Use the DISK1 routine to write a listing page heading containing EBCDIC blanks on the first sector of the CIB.
7. Make sure there are at least 33 sectors of Working Storage available.
8. Read in Phase 1 overlaying Phase 0.

LSS33: Less than 33 sectors of Working Storage. Branches to print an error message (A 01) and returns control to the Supervisor.

Phase 1 (Control Records)

Chart: BO

- Reads a source record.

- Processes Control Records.

- Sets switches for specified options.

- Prints control record read.

- Overlays Phase 1 with Phase 1A.

Operation

Phase 1 reads in and processes control records; the data on each control record is compared with data stored in core. When the data in the control record matches the string of data in core, a switch is set indicating the options specified. When a non-control record is encountered, Phase 1A is read in overlaying Phase 1.

S1A: Entry Point for Phase 1: Uses the read card (RDCRD) subroutine to read one card or paper tape

record. If the record is not a control record, go to ENDCC. XR1 is set to the first word of the input record. XR2 is set to the number of words in the string in core containing a control record. XR3 is set to the address of the first word of the string in core.

CKBLN (Check Blank): Check the input record (character by character) for blanks. If all 70 characters are blank, go to NXSTR. When a non-blank character is found, go to PSTBL.

PSTBL (Past Blank): Compare the input character with the character in the string stored in core. If they do not match, go to NXSTR. If they match, return to CKBLN to process the next character. If all characters in the input record match the characters in the string in core and a blank follows the last character, go to the routine servicing the record type. If the input record does not match any string in core or if the character following a matching record is not blank, go to NOCTL.

NXSTR (Next String): Update counters and index registers to scan the next string. If the input record has been checked with each string and no matching string is found, go to NOCTL.

INTCC: Initializes to scan all strings. This is done after one control record has been completed and before the next record is read.

COMN1: Go to CLCTN to compute size of COMMON. Save the size of COMMON in SCOMN and go to CCCOM.

DEFINE (Define File Size): Set FILSW indicating a FILE control record. Go to CLCTN to obtain the number of sectors required by the program at object time. Store the number of sectors at FILSZ (File Size). Increment ADCOW by 7. Note that when *FILE is used, the first data word that would normally have been assigned to relocatable address zero will now be assigned to relocatable address seven.

INTLV (Interrupt Level): Go to CLCTN to obtain the interrupt level number. Save the interrupt level number and decrement the interrupt level count by 1.

CLCTN: Used by DFINE, COMN1, and INTLV to obtain numeric information from control records. Sets up the SCAN routine to allow only the processing of numeric information and then use the SCAN routine to evaluate the number of sectors or the interrupt level number contained in the control record. The value will be returned in the accumulator. Any error will cause an exit to NOCTL.

PRTST (Print Symbol Table): Sets bit 0 in STOPT (Symbol Table option) switch when a PRINT SYMBOL TABLE control record is processed.

LIST: Set bit 0 in LSTOP (list option) switch when a LIST control record is processed.

PCHST (Punch Symbol Table): Sets bit 15 in STOPT switch when a PUNCH SYMBOL TABLE control record is processed.

TWOPS (Two Pass Mode): Sets PSMDE (Pass Mode) switch to zero when a TWO PASS MODE control record is processed.

LSTDK (List Deck): Sets bit 0 in LDKOP (List Deck option) when a LIST DECK control record is processed.

EDIT: Sets bit 15 in LDKOP when a LIST DECK E control card is processed.

SAVST (Save Symbol Table): Sets SAVSW (Save Symbol Table) switch on (non-zero) when a SAVE SYMBOL TABLE control record is encountered.

SYSTB (System Symbol Table): Reads the System Symbol Table into the Symbol Table area. The System Symbol Table resides in the Assembler area on the disk. The first word will be a count of the symbols in the System Symbol Table. This will be used to initialize the symbol count (CTSYM) and position the lower limit of the Symbol Table (LOEND).

CCCOM (Current Control Record Common): Use the principal printer to print the control record and go to INTCC to initialize for the next control record.

NOCTL (NOT Control Record): Insert ID into the input buffer before listing the control record. This record has an * in the first position indicating a control record, but was not recognizable or was an illegal Level or File record.

ENDCC (End Control Record): Read in Phase 1A.

Phase 1A

Chart: BP

- Move record right 20 positions if input is from paper tape.
- Initialize the Symbol Table limits according to the options specified.
- Read in Phase 2.

The current record is not a control record. Restore SCAN and go to CRDIO if input is from cards. If the input is from paper tape, move the current record over 20 positions and set the read-in address for position 21 of the I/O area.

CRDIO (Card I/O): If Pass 2, go to FTCH2; otherwise initialize for Symbol Table overflow. Compute the End of Symbol Table address (ENDST) on a basis of the options specified by the control records. Set up a word count of 320 and a sector address of 0 (relative to start of Working Storage) at ENDST-2 and ENDST-1 respectively. Go to FTCH2.

FTCH2 (Fetch Phase 2): Use the DISK1 routine to read in Phase 2 overlaying Phase 1.

Phase 2 (Header Statement Processing)

Chart: BQ

- Process ABS, ENT, ISS, ILS, EPR, SPR, LIBR, statements.
- Initiate reading of Phase 5.
- Initiate reading of Phase 6.
- Transfer control to Phase 9.

This routine is entered at STRT2 when entered from Phase 1A.

STRT2: Initialize ENTCT to allow 14 entry points. If Save Symbol Table option is in effect, set relocation mode (RLMDE) to 0 for absolute and allow only ABS and HDNG in Program Header group.

S2000: Bypass a comment record (* in postion 21).

S2003: Pack and save op code. Look up op code in small op code table. If op code not in table, go to SZOUT.

OPVC2: Transfer Vector DC Table for Program Header group. Sections of code for a particular op code are reached by an indexed, indirect branch, where an entry in the table becomes the effective address of the branch.

TB2ST: Beginning of small op code table. Includes following mnemonics: ABS, ENT, LIBR, ISS, EPR, SPR, and ILS.

ENT1: ENT processing. Assembly relocation mode must be relocatable.

S2006: Op code error code entered in buffer. This error will occur if mutually exclusive op codes of the Program Header group are in the same source program, i.e., ABS and ENT.

S2008: ENT must not be preceded by ISS or ILS. Up to 14 ENTs allowed. Each ENT increases word 6 of the Program Header (Length of Header -9) by three. In Pass 2, S2100 is used to collect the entry point name and to look up the address of the entry point.

ISS1: ISS processing. The relocation mode of the assembly must be relocatable. An ISS cannot be preceded by an ENT, ILS, or another ISS. Set up SCAN to allow only numeric operand. SCAN is then used to evaluate the ISS number in positions 32-33 of the ISS record. If Pass 2, S2100 is used to set up scan for symbolic operand and collect entry point and evaluate its address.

LIBR1: LIBR processing. Assembly relocation mode must be relocatable. LIBR not permitted if no entry point in source program (ENT, ILS, or ISS).

ABS1: ABS processing. Must not be preceded by LIBR, ENT, ILS, or ISS. RLMDE (Relocation mode) is set to 0 for absolute assembly, and the primary (ADCOW) location counter and secondary (ADCW2) location assignment counter are set equal to ADCNI (Resident Supervisor with DISKN).

ILS1: ILS processing. The relocation mode of the assembly must be relocatable. Only one ILS statement is permitted and it must not be preceded by an ENT or ISS statement. The interrupt level (positions 32 and 33 of the I/O buffer) is stored in ISSNO until the program header is constructed in Phase 12.

EPR1: EPR processing. Must not be preceded by SPR.

SPR1: SPR processing. Must not be preceded by DPR.

S2100: Subroutine used by ENT and ISS during Pass 2 to collect entry point name and evaluate its address. Since the pass mode (PSMDE) determines the object output buffer, S2100 will store the entry point name and address in the program header in DFBUF or BUFI for one pass mode or two pass mode, respectively. The name and address of the first entry point will also be stored in COMMA.

HDNGA: The HDNG op code is permitted anywhere in the source program. When Phase 2 is in core, Phase 5 must be read in to process the HDNG statement and then Phase 2 is restored. Further Phase 2 type op codes may then still be processed.

S2OUT: Phase 2 exit. Set up FLIPR to unconditionally read in Phase 6 and transfer to BGASM in Phase 9 to process the current source statement.

Phase 3

Chart: BR

- Save the Symbol Table (optional).
- Print the Symbol Table (optional).
- Punch the Symbol Table (optional).
- Set up FLIPR to read in Phase 4.

S3A: Phase 3 begins at this address. If the SAVE SYMBOL TABLE option was not selected, the Assembler branches to S3A2. If there were any assembly errors (ERCNT non-zero), the Symbol Table cannot be saved. If the number of entries in the Symbol Table exceeds the count contained in the constant at D100, the Symbol Table cannot be saved (STPSV set non-zero).

SV1: Set up the word count and sector address to save the Symbol Table. Insert the symbol count (CTSYM) as the first data word of the System Symbol Table (next word after the sector address). Temporarily set the file protect address to zero to allow the System Symbol Table to be written in the Assembler area on disk.

WRTST: Write the System Symbol Table and then restore the file protect address.

S3A2: Save the relative sector address of the disk sector of DSF output, and save the relocation mode of the assembly. If any condition causes an inhibit of the Symbol Table save (STPSV non-zero), use GETER to read the error print routine from Disk, and print error message (A 04).

S3A3: Go to S3OUT if no symbols in Symbol Table (CTSYM = 0). Move the Symbol Table that may reside in the area of the principal print routine and in Phase 11 to Phase 9.

LIPH3: Loop to make move described above. Set up the word count and sector address to read the principal I/O routine and the principal print routine from the disk. Go to RSTIO if print routine already in core (LIST option selected); otherwise, read principal print routine.

RSTIO: Read principal I/O routine. Use the PLNIO routine to blank the I/O buffer. If no Symbol Table output (print or punch), go to S3OUT. Go to NOPRT if PUNCH SYMBOL TABLE only. Use RPAGE to restore the page (printer), and then print a blank line (space). Move the words 'SYMBOL TABLE' to the I/O buffer (centered), and print. Print a blank line to provide a space after the title.

BLNIO: Subroutine to move eighty blanks into the I/O buffer.

NOPRT: Common point for print and punch. Start output of table at the high-core end.

L4: Use SUDMP to print and/or punch a record of five symbols. If the output is complete, go to S3OUT; if the output address (PARA1) has gone below the low-end address of the table (LOEND), go to DOSTO. If the output address is within 14 words of the breakpoint address caused when part of the Symbol Table was moved to Phase 9, the discontinuity must be corrected. If it is not within 14 words, go back to L4. In correcting the discontinuity caused by the move, the LOEND value will have to be changed to the address in Phase 9 of the last in-core symbol. The number of words in the discontinuity (one to fourteen) are moved to adjoin to the table moved to Phase 9. The table output now continues from the part of the table that is now in the Phase 9 area.

L5: Use SUDMP to output a record of five symbols. If there are any overflow sectors, go to L5A; if there are no more symbols, go to S3OUT; otherwise, go back to L5.

L5A: If the output address (PARA1) has gone below the value of LOEND, go to DOSTO. Go back to L5 if another complete record of five symbols can be outputted. Otherwise, set the temporary symbol count (TCONT) for the exact number of symbols left in the in-core table. SUDMP will then cause a record of fewer than five symbols to be outputted. Set LOEND to a large value before returning to L5. This will cause the test at L5A to go to DOSTO.

SUDMP: Subroutine to set up the conversion routine called DUMP, and to print and/or punch the record. If the punch option has been selected, go to S3PCH to read (if card). If printing the Symbol Table and the principal printing device is the 1132, restore page if printer on channel 12.

S3065: Set-up DUMP to output five symbols if more than five to go. Otherwise, set-up DUMP to do the exact number left. Use DUMP to convert to output format from symbol format.

S3TPR: Print output record and go to S3PC2 to punch the record if this option is also selected.

S3PC2: Punch the output record. Branch here for punch only, or after print if print and punch.

DOSTO: Output overflow sectors of table. Set TCONT equal to 106 for each sector of overflow.

L8: Read overflow sector.

L9: Use SUDMP to output a record of five. First time through the output address (PARA1) is at high-core end of overflow sector. If the sector is completed, decrement the overflow sector count (OFCNT) by one, and go to S3040 if overflow sectors remain. If sector is not complete, return to L9 to continue. When overflow sectors are completed, go to S3OUT.

S3040: Set-up for next overflow sector, and return to L8. Note that each overflow sector consisting of 106 symbols is outputted as 21 records of five symbols and one record of one symbol.

DUMP: Subroutine to convert from name code to EBCDIC. An M is inserted in front of each symbol that is multiply-defined, and an A is inserted in front of a symbol whose value is absolute in a relocatable assembly.

NOTE: The characters mentioned should not be considered to be Error Flag Indicators (see Table 3).

Phase 4

Chart: BS

- Print the number of errors in the assembly.
- Move the Disk System format output to the beginning of Working Storage.
- Return control to the Supervisor.

S4A: Start of Phase 4.

SPCE4: Space printer (or typewriter). Go to ERMSG if no assembly errors, and go to ONER if only one assembly error. Use routine starting at BC06 to convert the error count from binary to a sign and five decimal positions starting at OUTP4.

S4110: Move decimal error count into the output message string (MSG4).

ERMSG: Move error message into I/O buffer. When there are no assembly errors, the word 'NO' is used instead of an error count. Print message and go to S4A2.

ONER: Replace the S in 'ERRORS' by a blank, and move an EBCDIC one (1) into the error count position of the message. Go to ERMSG.

S4A2: Move the DSF output down four cylinders to the beginning of Working Storage. The number of sectors moved is rounded to the nearest number of half-cylinders of DSF output, since the move takes place by half-cylinders.

READD: Loop to read in one sector of DSF output from its sector position during the assembly, and then write it with its sector address reduced by 32 (four cylinders). When all sectors of DSF output have been moved in this fashion, the non-XEQ switch and the non-DUP switches are restored from TXQSW and TDPSW respectively. These two temporary values reflect the effect of assembly errors (if any) before returning to the Skeleton Supervisor.

Phase 5

Chart: BT, BU

- Processes OR, EQU, HDNG, BSS, and BES statements.

a. Sets the Location Assignment Counter equal to the operand value of an ORG statement.
b. Assigns the value of the operand to the label of an EQU statement.
c. Reserves a number of words in core equal to the operand value for a BSS or BES statement.

- Prints the heading when a channel 12 indicator is sensed.

ORGA: Branch table causes ORG processing to begin here. The SCAN routine is used to evaluate the operand, and then the ORGBS subroutine (common to ORG and BSS) is set-up for an ORG entry. ORGBS will under certain conditions insert an error code in position 18 of the I/O buffer. If error M (multiple definition), simply use first value to change the Location Assignment Counter. If any other error, ORG has no effect on the Location Assignment Counter.

ORGER: Use LDLBL routine to load label in Pass 1 (if any). The label of an ORG will have the value of the Location Assignment Counter before the ORG changed it. In Pass 2, the output options will be performed by going to LDLBL. Return to BGASM in Phase 9 to process the next statement.

BSSA: Branch table starts BSS (or BES) processing here. The label value (LABVL) is made even if odd when an E is present in position 32.

NALGN: The SCAN routine is used to evaluate the operand, and the ORGBS subroutine is set-up for a BSS entry. The value of the operand (number of words reserved by BSS or BES) is added to the label value and the sum is stored in the Location Assignment Counter. If BES statement (OPCNT--third word of op code table entry), the label value is set equal to the new value of the Location Assignment Counter (last reserved word plus one). The number of words reserved (Hex) is inserted into positions 9-12 of the I/O buffer. Exit at ORGER.

ORGBS: Common subroutine for BSS and ORG processing. If no error as a result of the operand scan, go to NOER. If error is present in Pass 2, go to ER2, otherwise use ERADD to enter the internal statement number (INTSN) in the 25 word error table (ERTBL).

ER2: If assembly is absolute, go to NOER2. In a relocatable assembly, an ORG operand must be relocatable, and a BSS operand must be absolute.

AB: Modified instruction. Conditional BSC tests for the condition tested in ER2. The condition codes are modified by ORG or BSS processing. Conditions are plus and minus for ORG (branch on condition zero) and zero for BSS (branch on condition non-zero). If operand relocation is valid, go to NOER2; otherwise insert an R (relocation error) in the I/O buffer and set the operand value to zero.

NOER: If Pass 1, go to ER2. In Pass 2 use ERSCH to determine if the internal statement number for this statement was added to ERTBL in Pass 1. If it was, insert a U (undefined error) in the I/O buffer.

NOER2: ORGBS exit.

EQUA: Branch table starts EQU processing here. Use SCAN to evaluate operand and store the value returned in the label value word (LABVL). If Pass 2, go to EQU2. If operand contained an undefined symbol, go to EQUER.

EQLBL: Save relocation value returned from SCAN in label relocation word (LABRL).

EQUER: Use ERADD to enter the internal statement for this statement in the error table.

EQU2: Use ERSCH to determine if the internal statement for this statement was added to the error table in Pass 1. If it was, insert U (undefined error) in the I/O buffer, and set the label value equal to zero.

EQUXT: Use LDLBL to enter label in Symbol Table or to do output options. Return to BGASM in Phase 9 to process next statement.

ERADD: Since any symbol in the operand of an ORG, BSS, BES, or EQU statement must be previously defined, a table is built during Pass 1, containing the internal statement numbers of the above statements whose operands were undefined. This table, known as ERTBL is 25 words long, and the address of the last entry is known as ERPTR.

ERSCH: This subroutine is used to search ERTBL. If the internal statement number is found in the table, a switch known as ERSW is set non-zero.

HD5: The GTHDG routine in Phase 9 reads Phase 5 into core and transfers to this label. An entry is made into the print routine at RPAGE to restore the page on the printer (dummy entry if print routine is for Console Printer). When the page restore is

finished, the heading is read into the print area (PAREA) from the first sector of the CIB.

L1: The binary page count (PGCNT) is converted to decimal and stored in positions 78-80 of the print buffer. Leading zeros in the page number are suppressed.

OUTVP: Increment the page count by one. Set up the print routine pseudo-transfer vector (TV3) to enter the print routine beyond the section which moves the I/O buffer to the print buffer. Print the heading line. Restore the pseudo-transfer vector for normal entry, and clear the page restore switch (EJECT). Restore the phase which was in core before GTHDG read in Phase 5. Return to GTHXT in Phase 9 to exit GTHDG routine.

HDNG5: If Pass 2, and LIST option selected, the HDNG statement is processed. The HDNG statement is punched (optional), and then the HDNG operand is centered, and written on the first sector of the CIB. If typewriter is print device, the heading line is typed, preceded, and followed by a line feed. If the 1132 is the print device, GTHDG is used to perform the new page routine. If HDNG5 entry is from Phase 2, Phase 2 is restored.

Phase 6

Chart: BV, BW

- Processes DC statements and all imperative instruction statements.
  a. Converts the operand of a DC statement to its binary value.
  b. Builds machine-language instructions for all imperative instruction statements.

INSTA: Branch table starts imperative instruction processing here. The five-bit op code obtained from the op code table is saved in the instruction buffer (INSBF). The tag position (33) is examined, and if it is blank, a branch is made to INST2. A non-blank tag is tested for validity. If it is not a 0, 1, 2, or 3, a T (tag error) is inserted in the I/O buffer. The tag bits, 6-7, are inserted in INSBF (zero if tag error).

INST2: The format position (32) is examined, and if it is blank or X a branch is made to SINST. If the long form of this instruction is not valid (controlled by third word of op code entry), go to FCER to insert F (format error) in I/O buffer and process as a short instruction. If format is I, go to IINST to process an indirect instruction, and if format is L, go to LINST to process a long instruction. Any other format is an error, but since instruction is valid in the long form, after error is inserted in the I/O buffer, Assembler branches to LINST to process as a long instruction.

SINST: If Pass 2, go to SI2ND; otherwise increment Location Assignment Counter by one and go to INSXT.

SI2ND: Save the Location Assignment Counter (TTEMP plus 2), because it must be incremented before going to SCAN and then restored before going to DFOUT. The Location Assignment Counter is then incremented by one (points to next instruction).

DISP: If instruction uses special operand (condition codes), go to SPOND. Use SCAN to evaluate operand. If format is non-blank, go to NOMOD (no modification). If op code is STX, go to MOD (displacement modification). If instruction is LDX, LDS, any shift instruction, or WAIT, go to NOMOD. For any other instruction, the tag bits are checked, and if they are zero, the Assembler branches to NOMOD.

MOD: Displacement modification. The value of the Location Assignment Counter is subtracted from the value of the operand and relocation value is made absolute.

NOMOD: No displacement modification. The displacement must be absolute or an R is inserted in the I/O buffer and the displacement is set equal to zero. The displacement must be in the range of minus 128 to plus 127 or an A (addressing error) is inserted in the I/O buffer and the displacement is set equal to zero.

INST3: The displacement is inserted in INSBF, and the Location Assignment Counter is restored from ITEMP plus 2. A zero (relocation code for the first word of a long instruction, or the code for a short instruction) is inserted in position 6 of the I/O buffer. INSBF is outputted in hexadecimal to positions 9-12 of the I/O buffer, and in binary to the object output buffer by DFOUT. The Location Assignment Counter is incremented by one, and if this is a long instruction, the Assembler goes to LI3RD (TWOSW non-zero).

INSXT: Uses LDLBL in Pass 1 to load label (if any), and in Pass 2 to do I/O options. Returns to BGASM in Phase 9 to process the next statement.

SPOND: Special (conditional) operand processing. This section checks each operand character for the condition codes shown below, and inserts the corresponding condition bits into INSBF for those it finds. It returns to INST3 when it reaches a blank, or when it detects an erroneous condition code (C inserted in I/O buffer). Shown below is a table of the condition codes and the bit each sets:

Bit Position Set	Condition Indicated
10	Zero (Z)
11	Minus (-)
12	Plus (+ or &)
13	Even (E)
14	Carry (C)
15	Overflow (O)

IINST: Indirect instruction. Inserts indirect addressing bit (8) in INSBF.

LINST: Long instruction. Inserts long instruction bit (5) in INSBF. If Pass 2, go to LI2ND; otherwise, increment Location Assignment Counter by one and return to SINST plus four to increment (ADCOW) for second word.

LI2ND: Turn on TWOSW to indicate a long instruction (for later use). Save Location Assignment Counter as in short instruction processing, and use SCAN to evaluate operand. Insert relocation property (0 or 1) in position 7 of the I/O buffer and save relocation bits with the operand value in ITEMP (2 words). If instruction may have condition codes (BSC), go to SPOND. Otherwise, return to DISP plus two to output first word of this instruction. Note that after the first word is outputted, if TWOSW is set, the Assembler will branch to LI3RD to output the second word.

LI3RD: Set TWOSW equal to zero to return to one-word mode. Output second word (saved in ITEMP) in hexadecimal to positions 13-16 of the I/O buffer, and use DFOUT to output the second word and its relocation indicator bits. Return to INSXT in the short instruction processing section.

DCA: Branch table starts DC statement processing here. If Pass 2, go to DC2ND; otherwise increment Location Assignment Counter by one and use LDLBL to load label (if any). Return to BGASM in Phase 9 to process the next statement.

DC2ND: Save the Location Assignment Counter in DCCN plus 1 before incrementing it by one (it must point to location of the DC plus one before entering SCAN). Use SCAN to evaluate operand and then restore the Location Assignment Counter from DCCN plus 1. Insert the relocation value of the constant in position 6 of the I/O buffer (0 or 1). Output the value of the constant in hexadecimal to positions 9-12 of the I/O buffer and use DFOUT to insert the binary value with its relocation indicator bits into the object output. Go back to DCA plus 4 to exit.

Phase 7

Charts: BX, BY, BZ, CA

- Processes DEC statements.
  a. Converts decimal integers to a 31-bit binary value.
  b. Converts fixed-point numbers to a 31-bit binary value.
  c. Converts floating point numbers to a 23-bit binary value plus an exponent.
- Processes XFLC statements.
  a. Converts floating point numbers to a 31-bit binary value plus an exponent.

DECA1: The branch table starts DEC processing here. If this statement is an XFLC, go to XFLCA. If the Location Assignment Counter value is now odd, go to ADJCT.

STOLB: Store the Location Assignment Counter in the label value. If Pass 2, go to DECIN; otherwise increment the Location Assignment Counter by two and go to DECCN-4 to exit to LDLBL.

ADJCT: Add one to the Location Assignment Counter to make it even. Go to STOLB to revise label value.

DECIN: Use FLOTD to evaluate DEC operand. Go to DECA if non-integer constant, and to DEFXP if integer. Treat integer as fixed point with a binary place value of 31.

DECA: If B-value specified, go to DEFXP. Otherwise form two-word constant in DECBF for output at DEOUT. (Convert negative constant to complement form.)

DEFXP: Fixed point section of DEC. Compute the shift count; go to FLERR if minus; if shift count is more than 31 go to FLZER. If sign of constant is plus, shift constant by the first shift count and go to

DEOUT. If sign of constant is minus, remove sign bit, shift right by shift constant, and convert to two's complement.

DEOUT: Output section of DEC. Insert relocation code (0) in positions 6-7. Output first word in hexadecimal to positions 9-12 of I/O buffer, and then use DFOUT to output in binary to Disk System format. Increment Location Assignment Counter by one and then output word two of constant in hexadecimal to positions 13-16 of I/O buffer. Use DFOUT to output word two in binary to DSF, and then increment Location Assignment Counter by one. Use LDLBL to load label (if any) in Pass 1 and to do I/O options in Pass 2. Return to BGASM in Phase 9 to process the next statement.

XFLCA: If Pass 2, go to XFLIN, otherwise increment the Location Assignment Counter by three and exit at DECCN-4 to LDLBL.

XFLIN: Use FLOTD to evaluate XFLC operand. If any B-value specified, go to FLERR. Convert magnitude and sign to complement form. Use DFOUT to insert the binary characteristic as the first word of the constant in the DSF, and then convert to hexadecimal and insert in positions 6-7 of the I/O buffer. Increment the Location Assignment Counter by one, and go to DEOUT plus 5 to output words two and three.

FLOTD – Floating Decimal: The FLOTD subroutine converts the operand of a decimal integer, fixed, or floating point number to their binary equivalents. A floating point number represented in powers of 10 will be converted to powers of 2. The FLOTD subroutine contains a scanning process which converts the operand to its binary equivalent and a post scanning process which converts from powers of 10 to powers of 2. Buffers FLE10 - BUF5 are initialized to zero upon entry to FLOTD; XR3, which is used to count digits to the right of the decimal point, is also set to zero.

Scanning Process: This portion of the FLOTD subroutine does the following:

1. Converts the decimal integer or the mantissa of a fixed or floating point number to its binary equivalent.
   (a) If the decimal integer or mantissa is negative, a /8000 is stored in the FLSGN buffer (the decimal integer or mantissa is processed as a positive number.
   (b) If a decimal point is included in the operand, a minus one is added to XR3 for each digit to the right of the decimal point (the operand is treated as an integer).
   (c) The binary value of the decimal integer or mantissa is stored into a 5-word buffer (BUF5) for further processing.
2. The value of a power of 10 exponent (E-type) is converted to its binary value and stored in FLE10.
3. The value of a binary point identifier (B-type) is converted to its binary value and stored in FLB2.

Assume an operand of 4.500 E-1; at the end of the scanning process the contents of buffers would be:

BUF5

Word 1	Word 2	Word 3	Word 4	Word 5
0000	0000	0000	0000	7194

XR3 = -3 FFFD

FLE10 = -1 FFFF

At the end of the scanning process, the power of 10 representation is:

(a) A binary mantissa representing an integer in BUF5.
(b) A binary power of 10 exponent (FLE10).
(c) A binary value equal to the number of digits to the right of the decimal point (XR3).

The objective of the post scanning process is:

(a) A binary fraction (left-justfied to the binary point identifier).
(b) A binary exponent using 128 as the zero point. Positive exponents will range from 129 (+1) to 255 (+127), and negative exponents will range from 127 (-1) to zero (-128).

The post scan processing is initialized to convert to a power of 2 by:

(1) Combining the values of XR3 and FLE10 to obtain the effective value of the power of 10 exponents. The result is stored in FLE10.
(2) Moving the binary point identifier (decimal point) from the end of word 5 to the end of word 2. This is effectively raising the power of 2 exponent +64.

(3) Set XR3 to represent the initial power of 2 exponent. This is 128 (zero) plus 64 (step b) or 192.
(4) Shift the mantissa left (normalized) until the high-order bit is in bit position zero of word 2. Decrement XR3 by 1 for each bit position shifted.

After initializing for the post scan processing, the buffers contain:

BUF5

Word 1	Word 2	Word 3	Word 4	Word 5
0000	8CA0	0000	0000	0000

XR3 = 192 `00C0`

FLE10 = -4 `FFFC`

The following steps are performed, during the post-scan process, to convert from a power of 10 to a power of 2.

1. Reduce the power of 10 exponent (FLE10) toward zero by:
   (a) Dividing the mantissa by 10 if the exponent is negative and multiplying the mantissa by 10 if the exponent is positive.
   (b) Add a 1 to FLE10 for each division and subtract a 1 from FLE10 for each multiplication.
2. Normalize the mantissa by shifting the high-order bit to bit position 0 of word 2.
3. Determine the effective power of 2 exponent by:
   (a) Adding a 1 to XR3 for each bit position shifted to the right.
   (b) Subtracting one for each bit position shifted to the left.
4. Repeat steps 1 to 3 until FLE10 is equal to zero. At the end of the post-scan process the buffers show:

BUF5

Word 1	Word 2	Word 3	Word 4	Word 5
0000	9000	0000	0000	0000

XR3 = 131 `0083`

FLE10 `0000`

Prior to returning to the DEC or XFLC routines, the contents of word 2 and word 3 are loaded into the accumulator and extension, and shifted right one to clear bit position 0 of the accumulator before performing an OR of the sign bit. The mantissa and sign are then stored in the FLBMN (Binary Mantissa) buffer.

The contents of XR3 are stored in the FLBCH (binary characteristic) buffer.

The output of the FLOTD subroutine is:

FLBMN (Mantissa) `4800` `0000`

FLBCH (Binary characteristic) `0083`

The scanning portion of the FLOTD subroutine is entered at FLOTD from the DEC or XFLC statement processing routines.

FLOTD — Initializes subroutine.
(a) Resets buffers and switches.
(b) Sets FLSGN equal to /8000 if the mantissa is negative.

FLLP — Converts the mantissa to its binary value and stores the binary value in BUF5.
(a) Checks each digit to determine if it is numeric and if it is not, goes to FLSSC.
(b) Converts character by character, beginning with the high-order digit.
(c) If a decimal point is encountered, add a -1 to XR3 for every digit to the right of the decimal point. (The instruction at FLLP will be changed from a NOP to an ADD by the FLSSC routine.)
(d) If BUF5 overflows indicating the mantissa is too large, go to FLERR.

FLSSC — Analyzes non-numeric operand characters.
(a) Branch to FLBSC if the character is a B (binary point identifier).
(b) Branch to FLESC if the character is an E (power of 10 indicator).
(c) Modify the instruction at FLLP2 to ADD if the character is a decimal point.
(d) Branch to FLFIN when a blank is found.
(e) Branch to FLERR if the character is not one of the above.

FLBSC — Initializes for the processing of B-type exponents.
(a) Set FLNIS to non-zero.
(b) Load the address of FLB2 to FL3+1.
(c) Go to step (c) of FLESC.

FLESC — Initializes for the processing of E-type exponents and processes E and B-type exponents.
(a) Set FLNIS to non-zero.
(b) Load the address of FLE10 into FL3+1.
(c) Modify the instruction at FL4 to ADD if the exponent is positive and to SUBTRACT if the exponent is negative.
(d) Go to FL2.

FL2 — Converts exponents to their binary value.
(a) If the exponent is an E-type, convert it to its binary value and store in FLE10.
(b) If the exponent is a B-type, convert it to its binary value and store it in FLB2.
(c) Exit to FLSSC when a character other than numeric is found.
The post-scan processing is entered at FLFIN from FLSSC of the scanning process.

FLFIN — Initializes for the post-scan processing.
(a) Add XR3 and FLE10 and store in FLE10.
(b) Load XR3 with 192 (128 + 64).
(c) Check BUF5 for a zero condition and branch to FLZER if zero or FLFNL is not zero.

FLFNL — This routine determines the direction of shift and if necessary shifts the mantissa right.
(a) Check word 1 of BUF 5; if zero, go to FLFNX.
(b) Branch to the SRT subroutine to shift the entire contents of BUF5 one position to the right.
(c) Add a 1 to XR3.
(d) Repeat steps (b) and (c) until word 1 is zero.

FLFNX — This routine determines if word 2 of BUF5 is negative and, if necessary, shifts mantissa left.
(a) Check word 2 of BUF5; if negative, go to step (c).
(b) Branch to the SLT subroutine to shift the entire contents of BUF5 one position to the left.
(c) Subtract 1 from XR3.
(d) Repeat steps (b) and (c) until word 2 is negative.
(e) Check the value of FLE10 and if negative or zero go to step (h).
(f) Subtract 1 from FLE10.
(g) Branch to the multiply (MPY) subroutine to multiply the contents of BUF5 by 10 and return to FLFNL.
(h) Check FLE10 and branch to FLFEX if zero.
(i) Add a 1 to FLE10.
(j) Branch to the Divide (DIV) subroutine to divide the contents of BUF5 by 10 and return to FLFNL.

FLFEX — This routine stores the mantissa and exponent into buffers to be used by the DEC or XFLC routines.
(a) Store the contents of XR3 into FLBCH (binary characteristic).
(b) Load words 2 and 3 of BUF5 into the accumulator and extension. Shift right one and insert the mantissa sign bit. (FLSGN contains a /8000 if the mantissa is negative.) Store in FLBMN (binary mantissa).
(c) Check the binary exponent to determine if it is greater than 256 or less than 0. If it is, branch to FLERR. If not, go to FLXXX.

FLXXX — Load XR2 with the address of FLBMN. Exit via the return address at FLOTD.

FLZER — Floating zero routine.
(a) Clear buffers and switches.
(b) Set data in FLNIS.
(c) Go to FLXXX.

FLERR — ERROR Routine.
(a) Load S (syntax error) into position 18 or 19 of the I/O buffer.
(b) Go to FLZER.

Phase 8

Charts: CB, CC, CD

- Processes CALL, LIBF, DSA, LINK, EXIT, and EBC statements.
  a. Converts the operand (subroutine name) to name code for CALL and LIBF statements.

b. Reserves three words in the program (these will be filled by the Loader) for a DSA.
c. Generates four words in the object program. Words 1 and 2 are a long BSI to MONCL +1. Words 3 and 4 are the program name in name code for a LINK.
d. Generates a short LDX, tag 0, to MONCL for an EXIT statement.
e. Reserves the needed storage for the operand of EBC.

LIBFA Statement: Beginning of LIBF processing. This label is reached from CALL processing on the basis of information contained in the third word of the op code table entry. The relocation bits (in hexadecimal) are 20 and are saved in INDBT. If Pass 2, go to CA2ND; otherwise, increment the Location Assignment Counter by one and go to CLLXT to exit.

CALLA: Branch table starts CALL processing at this point. If LIBF (see above), go to LIBFA. Set relocation indicator bits (in hexadecimal) to 30 and save in INDBT.

CALLC: If Pass 2, go to CA2ND; otherwise, increment the Location Assignment Counter by two and go to CLIXT to exit.

CA2ND: Use CLLCT subroutine to collect the name of the call. The name is returned in the accumulator and extension and, if it is blank (accumulator equal zero), the Location Assignment Counter is incremented by one (LIBF) or two (CALL) before going to CLLXT to exit.

COP: CALL (or LIBF) output. The relocation code (3 or 2) is inserted in position 6 of the I/O buffer for word one of the call name. The first word of the name (in hexadecimal) is inserted in positions 9-12, and DFOUT is used to enter word one (in binary) with it relocation indicator bits into the DSF. The Location Assignment Counter is incremented by one if this is a CALL statement. A zero is inserted into position 7 of the I/O buffer, and word two of the name is inserted into positions 13-16. DFOUT is used to insert word two of the name into the DSF. The Location Assignment Counter is then incremented by one.

CLLXT: Use LDLBL to load label (if any) during Pass 1, and to do output options during Pass 2. Return to BGASM in Phase 9 to process the next statement.

DSAA: Branch table starts DSA processing here. If Pass 2, go to DS2ND, otherwise increment the Location Assignment Counter by three, and go to DSA2-4 to exit.

DS2ND: Relocation code for word one is three and for word two, one. Use CLLCT routine to collect the name. Go to DSA2 if the name is all right; otherwise, increment the Location Assignment Counter by three and go to DSA2-4 to exit.

DSA2: Insert 3, relocation code for word one of name, into position 6 of the I/O buffer. Output the first word of the name (in hexadecimal) to positions 9-12, and use DFOUT to insert word one (in binary) with its relocation indicator bits into the DSF. Increment the Location Assignment Counter by one. Insert a 1, relocation code for word two of the name, into position 7, and insert word two of the name into positions 13-16. Use DFOUT to output word two and its relocation indicator bits to the DSF. Increment the Location Assignment Counter by two before exiting. Note that the DSA statement will generate a data header in the DSF since only two words are actually output.

LINKA: Branch table starts LINK processing here. If Pass 2, go to LK2ND; otherwise increment the Location Assignment Counter by four before going to DSA-4 to exit.

LK2ND: Use CLLCT routine to collect the name of the program link. If the name is all right, go to LINK2; otherwise, increment the Location Assignment Counter by four, and go to DSA-4 to exit.

LINK2: Insert a zero into position 6 of the I/O buffer, and insert word one (first word of a long BSI--4400 in hexadecimal) into positions 9-12. Use DFOUT to enter this word (in binary) into DSF, and then increment the Location Assignment Counter by one. Insert a zero into position 7, and then insert word two (address of MONCL plus one) into positions 13-16. Use DFOUT to enter this word into DSF, and then increment ADCOW by one. Use DFOUT to enter word three (first word of link name) into DSF, and then increment ADCOW by one. Use DFOUT to enter word four (second word of link name) into DSF, and then increment ADCOW by one. Go to DSA-4 to exit.

EXITA: Branch table starts EXIT processing here. If Pass 2, go to EX2ND, otherwise increment ADCOW by one and go to DSA-4 to exit.

EX2ND: Insert a zero into position 6 of the I/O buffer, and insert a short LDX instruction, tag 0, to MONCL into positions 9-12. Use DFOUT to insert this instruction into DSF, and then increment ADCOW by one. Go to DSA-4 to exit.

EBCA: Branch table starts EBC processing here. Position 35 is checked for the presence of the period (.) delimiter. If it is present, go to EBLP-1. If it is missing, an S (syntax error) is entered in the I/O buffer, the Location Assignment Counter is incremented by 18 (maximum number of words generated by an EBC), and the Assembler exits at EBX.

EBLP: The operand field is scanned from position 71 to the left for the right-end delimiter. When found, go to EBDL.

EBDL: A blank is used to replace the delimiter just detected. Then if the character count between the delimiters is odd, the blank for the right half of the last compressed word will be present. The number of characters is stored in EBBF and adjusted to be the next even number if odd before dividing by two to obtain the word count. The number of characters (EBBF) is inserted (in hexadecimal) into positions 9-12 of the I/O buffer.

EBXR2: The number of words to be generated by this EBC is stored into the second word of this LDX instruction.

EBPCK: Loop used to pack the characters in the EBC operand, two characters per word, and insert into DSF by means of the DFOUT subroutine during Pass 2. Increment the Location Assignment Counter by one for each time through the loop, in Pass 1 and in Pass 2.

EBX: Use LDLBL to load label (if any) in Pass 1, and to do I/O options in Pass 2. Return to BGASM in Phase 9 to process the next statement.

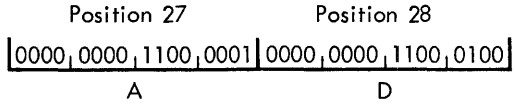
Phase 9. (Non-Overlaid Mainline)

Charts: CE, CF, CG, CH, CI, CJ, CK, CL, CM, CN, CO

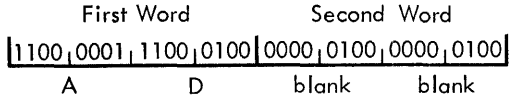
- Searches the op code table to ascertain valid operation mnemonic.
- Transfers control to the proper statement processing routine (directly or through an overlay read-in routine).
- Contains subroutines common to statement processing.

BGASM: Entry point for Phase 9. If the record is a comment record, do not process. Pack the mnemonic op code into two words and save in the OPBUF (Op Code Buffer).

Input-Buffer:



Op Buffer:



Representation of the mnemonic AD (Add Double)

OPLP: Performs a search of the op code table (BEGOP) using the op buffer contents as the argument.

The search is performed using XR3 as the entry pointer. When an equal entry is found, a branch to the OPOK routine is executed. If an equal entry is not found, an O (Op code error) is placed in position 17 or 18 of the I/O buffer and the program exits to the LDLBL routine if in Pass 1; otherwise, two zero words are inserted in object output before going to LDLBL.

OPOK	Using XR3, the third word of the op code table entry is obtained (see Operation Code Table). Bits 12-15 are used as a transfer vector for transfer of control through a branch table to the individual routine used in processing the statement, or to a routine which reads in the proper overlay and repeats the transfer vector branch to the branch table.
COMPT	Initializes the FLIPR routine to read in the phase required to process the statement.

Scan: This subroutine converts the expression in the operand field to its binary value. It also determines whether the operand expression is absolute or relocatable.

SCAN	Entry point for the scan routine. Initializes the scan routine.

SCNLP (Scan loop) Checks the operand character to determine if it is an operator (+, -, or *) or delimiter (blank or comma) and branches to the routine that processes the special character.

TYPE Checks the operand character to determine the type of operand element to be processed and branches to the routine servicing the element. The instruction at TYPE is modified to go directly to the servicing routine once the type is determined. It will be changed to a N0P instruction when an operator or delimiter is encountered.

SKPCB This routine leaves the scan pointer (XR1) pointing at the terminator (if blank) or one character position past the terminator (if comma).

FERR (Format error) Clears buffers and enters an S (syntax error) into the I/O buffer when an operand error is detected.

SCNEX (Scan exit) Uses the SKPCB routine to set up for processing the second operand. Exits to the Scan return address.

SMBOL Initializes to process symbols and returns to SCNLP.

SYMTP Modifies TYPE to return to SYMTP. Exits to GENRT+2 if the symbol has five or less characters and to FERR if it has more than five characters.

DCINT (Decimal integer) Initializes the INTYP routine to collect decimal integers. Modifies TYPE to branch to INTYP. Returns to SCNLP.

HXINT (Hexadecimal integer) Initializes the INTYP routine to collect hexadecimal integers. Modifies TYPE to branch to INTYP. Returns to SCNLP.

INTYP (Integer type) Converts decimal or hexadecimal integers to their binary equivalents. Returns to SCNLP.

CHRVL (Character value) Collects character values. Modifies TYPE to branch to FERR if there is more than one character in a character value.

STRX (Asterisk) Determines if the asterisk is an operator or an element. Branches to STAR if it is an element. Modifies the CON routine to multiply the previous operand element by the one following the asterisk and to suspend addition or subtraction until all multiplication has been completed. Goes to GENRT.

STAR Sets RLSCW (relocation switch) equal to the mode of assembly (RLMDE). Assigns the asterisk the current value of the Location Assignment Counter. Goes to CHRVL+2 (ensures an operator follows the asterisk).

SPLUS Branches to the COLL routine to perform the arithmetic operation which was determined by the previous operator. Modifies the COLL routine to add the next value of the next operand element to the VALUE buffer.

SMNUS Same as SPLUS except it modifies the COLL routine to subtract the value of the next operand element from the VALUE buffer.

SCMMA Branches to FERR if the comma is in the operand field of a short instruction. Goes to SBLNK.

SBLNK Branches to the COLL routine to process the last operand element. Branches to RELER if the operand is neither absolute or relocatable. Normal exit is to SCNEX.

RELER (Relocation error) Enters an R into the I/O buffer using the FERR routine.

UDFER (Undefined Symbol Error) Enters a U into the I/O buffer if a symbol in the operand is not in the Symbol Table.

COLL (Collect) Entered when an operator or terminator is found in the operand. This routine performs the arithmetic specified by the operators in the operand expression. The result is stored in the VALUE buffer. It also resets the Scan routine after each operand element has been processed.

LABCK This subroutine checks labels for validity and converts them into name code. LABCK is called by the LDLBL (Load Label) routine when building the Symbol Table and from SCAN when determining the value of an operand field containing labels.

The input is a label (five words; one EBCDIC character per word) from the label field of the I/O buffer or from the operand field.

LABCK first checks the validity of the label. A label is valid if:

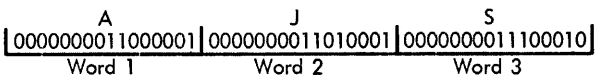
1. The first character is not numeric.
2. There are no blanks between characters.

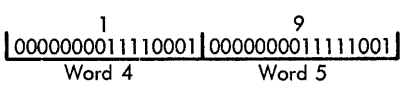
3. All characters are in the range of A-I, J-R, S-Z, 0-9, or #, @, or $.

If the label is not valid, an L (Label error) is inserted in position 18 or 19 of the I/O buffer and the label is ignored (zeros are returned to the calling routine).

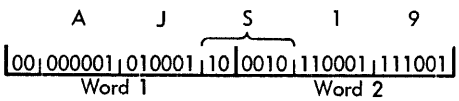
If the label is valid, it is converted to name code. The label AJS 19 is compressed as follows:

Label (positions 21-26 of the I/O buffer)





Symbol Table entry (name code)



Hex:

01462C79

A valid label is placed in A and Q; zeros are returned if label is invalid.

STSCH — Symbol Table Search: This subroutine is entered from SCAN to look up the value of the symbolic operand just collected. It searches the Symbol Table (including any disk overflow) and returns the symbol value if the symbol is found. When a symbol is found, STSCH exits to the return address plus one. If the symbol is not found, STSCH exits to the normal return address. This exit to SCAN causes a U (Undefined error) to be inserted in the I/O buffer.

The following is a description of STSCH by label:

STSCH — Entry point. A check is made to determine if there is any table overflow. If no overflow, go to STS1A, otherwise save the partial sector (any symbols in the one sector I/O area of the table).

STS1A — Save index register 1. Use SRCH subroutine to search in-core table. If symbol is found at this point, go to STS1, otherwise go to STSXT plus 2 for possible overflow search.

STS1 — Symbol is found, therefore increment link word for return plus 1 to SCAN. Save the symbol value returned from SRCH, restore the partial sector previously saved if there is any table overflow.

STXR1 — Restore XR1 as per saved upon entry. This instruction is transferred to for an undefined symbol. The link word is not incremented.

STSXT — Symbol Table search exit. If a symbol is not found in the in-core table, transfer is made to STSXT plus 2. At this point, a check is made to determine if there is any table overflow. If there is none, the symbol is undefined, and transfer is made to STXR1. Otherwise, go to OFSCH to set up search of overflow sector(s).

GTOF2 — Use RDSYT subroutine to read overflow sector, and then use SRCH to search the one sector table. If found, go to STS1; otherwise, increment sector address of overflow and, if there are more overflow sectors to search, go to GTOF2. If there are no more overflow sectors, the symbol is undefined and transfer is made to STS2 to restore the partial sector.

SRCH — Search: This subroutine is used to do the actual search of the Symbol Table. The search technique used is a binary search. SRCH is entered from STSCH and STADD (Symbol Table Add). If SRCH finds a symbol, the return link word is incremented by one. The relocation and multiple definition bits (if any) are stored in a location named BITS. The limits of the table to be searched may be controlled externally, thus facilitating the use of SRCH for the in-core table and the one-sector table. The following is a description of SRCH by label:

SRCH — Entry point. Begin computing limits of search.

GO — Begin finding midpoint of table. Check if HI - LOW is greater than three. If not, search is done.

MID	Load XR1 with address of table midpoint. Compare input symbol (SYMBL) with this table entry. If not equal, go to NOTEQ. If symbol was found and SRCH was entered from STADD, this is a multiply-defined symbol. If entry was from STSCH, save relocation and multiple definition bits, increment link word, and exit.
NOTEQ	Input symbol was not equal. If input symbol was greater, (higher in alphanumeric sequence) go to SECHF. If not, the last midpoint address is the new LOW address.
SECHF	Input symbol was greater. The last midpoint address is the new HI address.
MULT	Multiply-defined symbol. OR the multiple definition bit into the table entry just found. Transfer to NOTEQ-6 to increment link word, clear ADDSW (indicates entry from STADD), and exit.
SAREA	Source Area. 80-word buffer for card or paper tape.
ERTBL	Table of internal statement numbers of EQUs, ORGs, and BSSs whose operands where undefined in Pass 1.
BTHEX	Binary to hexadecimal (General). Input is binary word in accumulator. Output is 1-4 hexadecimal EBCDIC characters starting in address specified in XR3.
B4HEX	Binary to 4 hexadecimal characters. Uses BTHEX to output exactly 4 hexadecimal characters.
ERFLG	Error Flag. Inserts error codes into positions 18 and 19 of I/O buffer (SAREA). Each entry also increments error count (ERCNT) by 1. NOTE: ERCNT is the count of the total number of errors, not just those reflected in positions 18 and 19.
GTHDG	If listing is specified, and 1132 is principal printer, read Phase 5 into core and transfer control to new page routine (HD5). After new page routine is performed, control is returned here, and the overlay that was in core before Phase 5 is restored.
LDLBL	Load Label. Uses LABCK (Label Check) to collect label. If label OK, goes to STADD in Pass 1.
STLBL	Label value to positions 1-4 of SAREA in hexadecimal.
PALBL	Pass label. Secondary load label entrance. Statements whose label fields are ignored enter here.
A1	If output (object or intermediate) fills Working Storage, print error message (A03). If Pass 1, set PSMDE = 0 for two pass assembly. The Assembler will stop at a WAIT instruction after printing the error message. This will enable the operator to take the necessary action with regard to his card or paper tape input. The assembly will continue in the two pass mode after pressing Start. When it is undesirable to alter the JOB source, the Assembler will eventually trap out the next Monitor control record and pass it to the Supervisor. If Pass 2, exit to Supervisor (MONCL). If Working Storage not yet full, save source statement in intermediate output if one pass assembly (INT1).
RDSRC	Read Source record and exit LDLBL.
AITW0	If no LIST, go to PUNCH. If Console Printer is print device, go to PRLNE and print record. If 1132 is print device, go to GTHDG if this is the first source record, or if printer is on channel 12.
PUNCH	No LIST DECK unless two pass assembly. If not, get next record from intermediate input (INT2) and exit.
PCHDK	Go to RDSRC if LIST DECK or LIST DECK E options not selected. Go to ERS if LIST DECK E.
PCHNB	For full list deck, find end of object output (1-19) and set punch count for this number.
ERS	Punch errors only. Go to RDSRC and read next record if no errors. Otherwise, blank positions 1-17 and set punch count to 20 before punching. Note that when punching paper tape the punch count is not needed, since the object and source is punched into the tape.
GETER	Reads error printing routine into BUFI from disk.

RDSYT – Read Symbol Table: This subroutine is used by STSCH and STADD to read one sector of Symbol Table overflow.

RSTRE – Restore: This subroutine is used by STSCH and SRCH to restore full table status after an overflow search. It uses RDSYT to read in the partial sector initially saved, and resets the HIEND and LOEND addresses.

OFSCH – Overflow Search: This subroutine is used by STSCH and STADD whenever it is necessary to initialize for an overflow search. The LOEND address is saved in SVLOW, the overflow sector address is set for the first overflow sector, and the HIEND and LOEND addresses are set up for a one-sector table. Upon exit, XR3 contains the number of overflow sectors.

WRSYT – Write Symbol Table: This subroutine is used by STSCH to save a partial sector and by STADD to either save a partial sector or to write a complete sector of table overflow.

DFOUT – Disk Format Output: This subroutine is used in Pass 2 to build the object output sectors in Disk System format (DSF). Data is input to DFOUT, one word at a time, along with the two relocation indicator bits which pertain to this word. The data word and its indicator bits are stored in a two-word buffer named TRWRD. DFOUT also checks for data breaks by comparing the primary Location Assignment Counter (ADCOW) with the secondary Location Assignment Counter (ADCW2). If they are not the same, DFOUT transfers to the data header subroutine (DTHDR) to insert a data header in the DSF. After each data word has been output, DFOUT transfers to the write disk format output subroutine (WRDFO) to determine if the one-sector buffer is full. The following is a description of DFOUT by label:

DFOUT — Entry point. If first entry, store load address in first data header (follows immediately after Program Header Record). Set ADCW2 equal to ADCOW.

CMPCT — Compare the Location Assignment Counter (ADCOW) with the secondary Location Assignment Counter (ADCW2). If equal, go to DFXR1. Otherwise, go to DTHR unless a data header was just generated because of a new sector (see WRDFO).

FXDTH — Fix data header. If a break in sequence occurs, and a new sector data header was just generated, the Location Assignment Counter (current) is stored in the first word of this new data header.

DFXR2 — Disk format index register 2. Index register 2 points to the buffer word containing the relocation indicator bits for the current block of eight data words. If first entry, or if starting a new block of eight words, XR2, XR1 (points to buffer word where this data word will be stored), and XR3 (shift count for indicator bits) are initialized.

DFXR3 — Disk format index register 3. Load shift count to XR3. Store data word and increment program word count (WRDCT) by one. Shift data word's indicator bits and OR to indicator bits word (XR2). If this data word is the first word of an LIBF name, ADCW2 is not incremented, since the two words of the name will be replaced by a short BSI at load time.

DFOXT — Disk format exit. Clear two-word input buffer and redundant data header switch (RDTHD) to zero.

LDXRS — Load index registers. Go to WRDFO to check if output sector is done. Reload index registers as per saved upon entering. Exit.

STADD – Symbol Table Add: This section of Phase 9 is used only in Pass 1 to build the Symbol Table; consequently it is overlaid during Pass 1 processing of the END statement by Phase 10 (consisting of the DTHR and WRDFO subroutines).

If label field of a statement is blank, STADD transfers to ADDXT which exits to load label subroutine. SRCH is used to determine if the label has already been defined in the in-core table. If it has not, and there is no table overflow, go to STAD2 to add label to table. If defined in in-core table, go to ADDXT. If not defined in in-core table and there are overflow sectors, this overflow must be searched. WRSYT is used to save the partial sector, and then OFSCH is used to initialize the overflow search. In STADD, all sectors of overflow must be searched to detect a possible multiple definition. If a multiply-defined symbol is found on an overflow sector, this sector is rewritten so as to contain the multiple definition flag bit. If the symbol is not multiply-defined on any overflow sector, the full table is first restored before making the new entry.

The following is a description of STADD by important labels:

STAD2 — Label not multiply-defined and can be added to table. If this label is the next higher alphameric entry (no table move

required), go to SYMIN for direct add to table. Otherwise move all higher entries three words toward lower core (one entry leftwards).

SYMIN — OR label relocation value (LABRL) into label before adding to table. Increment the count of number of symbols (CTSYM) by one. Reduce LOEND address by three, and determine if in-core table full. If not, go to ADDXT. Write overflow sector, increment count of overflow sectors (OFCNT) by one, and check whether or not the four cylinder overflow area has been exhausted. If not go to ST002. If it has, print error message (A02) and return to Supervisor.

ST002 — Increment sector address of table overflow, and reset table limits to allow another sector of overflow (106 more symbols).

INT1 – Intermediate I/O Pass 1: This subroutine is used in Pass 1 of a one pass assembly to save the source input on the disk. It is overlaid by INT2 during Pass 1 processing of the END statement of a one pass assembly. Each source statement is packed, two characters per word, and preceded by one word referred to as a prefix word. The prefix word contains the word count of the statement (including the prefix word) in bits 8 - 15. Bit 0 is used to indicate that the statement begins in position 21. Bit 1 is used to indicate that the statement includes an ID field.

The following is a description of INT1 by principal label:

INT1 — Entry point XR1 points to the prefix word for this statement, XR2 points to position 21 of I/O buffer (SAREA), and XR3 is set to five to check the label field. If there is any non-blank in the label field, go to IN105. When there is no label field, packing starts at position 27 (LFT27).

IN105 — OR bit 0 (CON plus 2) to prefix word and start packing at position 21.

BLSCN — Scan from position 71 toward left pointer (position 21 or 27) for a non-blank. If a non-blank found, go to LVSCN.

LVSCN — Compute number of positions from left pointer to end of statement. If minus (blank record), go to LDCTL. Add two (CON plus 5) to this number and divide by two. This result is the number of words to be packed (also enters prefix word).

LFPTR — Pack statement, two characters per word, and store in output buffer (BUFI).

NOPBR — NOP or branch. When INT1 is entered, this word is set to an NOP to allow checking for the ID field. If the ID field is found, this word is set to a branch to LDCTL. This allows the same packing routine to be used for the ID field (LFPTR). If there is no ID field, go to LDCTL.

ID — ID field found. OR bit 1 (CON plus 4) to prefix word. Add four to word count in prefix word. Use packing routine at LFPTR to pack ID field.

LDCTL — Reset NOPBR to a NOP. Check if room in buffer for one more statement (use maximum statement length). If not enough room, go to WRTIO.

NOWRT — CTLWD contains address in buffer of next prefix word. Set this next prefix word equal to 0001 (minimum word count). Exit.

WRTIO — Compute total word count for this sector (including prefix words). This count becomes the first data word in this sector.

CTLX2 — Write sector of intermediate I/O. Decrement count of sectors of Working Storage left (SCRA) by one, and increment sector address of intermediate I/O by one. Go back to NOWRT to initialize before exit.

Phase 10

Chart: CP

- Enters a data header in DSF when required (DTHDR).
- Writes one sector on the disk when the output buffer is full (WRDFO).

DTHDR (Data Header): This subroutine enters a data header in DSF as required. The program word count (WRDCT) is incremented by two for each entry to this subroutine, since each data header is two words long. The number of words to and including this new data header is stored in word 2 of the last data header. The Location Assignment Counters (ADCOW and ADCW2) are set equal, and the load

address (ADCOW) is stored in word 1 of the new data header. The address constant which is used for determining the number of words between data headers (DHPTR) is set to the address of the first word of this new data header. The buffer address where the next data word is to be stored (DFXR1 plus 1) is incremented by two. XR2 and XR3 initialization is also performed. The number of words to be moved, in the event that this data header is a normal new sector data header, is incremented by one so that all data words will be moved (see WRDFO). Exit.

WRDFO (Write Disk Format Output): This subroutine writes one sector of Disk System format output when the output buffer is full. After the sector is written, it moves any words past the three hundred and twentieth word of the buffer back to the beginning of the buffer. The following is a description of WRDFO by principal label.

WRDFO: Entry point. If the entry is from END statement processing, go directly to WRTSR to write the sector. Otherwise, check if output is past sector point of buffer. If output is past the three hundred and twentieth word of the buffer and a block of eight data words has been completed, go to DTHDR to insert a data header for the next sector and turn the redundant data header switch (RDTHD) before writing the sector. If the above condition does not exist, exit.

WRTSR: Write DSF sector. Increment sector address of DSF output by one, and decrement number of sectors of Working Storage available (SCRA) by one.

MVLFT: If entry is from END statement processing, no move is required. Exit.

MVCNT: The number of data words to be moved is contained in the second word of this LDX instruction. This number is a result of the difference computed when checking for output past the three hundred and twentieth word of the buffer, and is incremented in DTHDR by one to give the correct move number. If the count is zero, go to NOMVE.

W2: Loop to move words past the three hundred and twentieth word of the buffer back to beginning of buffer.

NOMVE: Initializes XR1 and XR2 control in DFOUT, and resets the data header address constant (DHPTR). Exit.

NOTE: In both DTHDR and WRDFO, reference is made to a buffer. This buffer will be DFBUF if assembly is in one pass mode, or will be BUFI if assembly is in two pass mode (see Figure 11). Both buffers are long enough to contain one sector of output plus a maximum of ten words past the three hundred and twentieth word. Note that various addresses in DFOUT, DTHDR, and WRDFO are set initially for DFBUF (one pass mode). Should the assembly be in two pass mode, these addresses will be set to their equivalent positions for BUFI during the processing of the END statement (Phase 12).

Phase 11

Chart: CQ

- Reads the source statements from the disk during Pass 2. (One pass mode assembly.)

INT2 (Intermediate I/O Pass 2): This subroutine is used in Pass 2 of a one pass assembly to bring the source statements saved by INT1 back into core in such a way that they are indistinguishable from statements read from the principal I/O device (card or paper tape). The following is a description of INT2 by principal label.

INT2: Entry point: Store blanks in I/O buffer (SAREA).

INT2A: Load XR1 with the next prefix word, and XR2 with the address of position 21. If prefix word indicates no label field, change XR2 for position 27.

LDR3: The second word of this LDX instruction contains the number of words to be unpacked (obtained from prefix word). If statement has an ID field, reduce word count by four. If the word count of this statement is 1 (only a prefix word, i.e., a blank record), go to NOID2.

UNPCK: Unpack statement into I/O buffer. Reduce the total data word count of this sector (contained in BUFI plus 2) by one for each time through this loop.

TSTID: Check prefix word to determine if this statement has an ID field. If no ID field, go to NOID2. Otherwise, go back to UNPCK and unpack four-word ID field.

NOID2: Check total data word count to see if this sector is exhausted. If it is, go to RDIO to read in next sector. Pack and save op code (save in OPBUF), and set XR2 for two positions preceding I/O buffer. Then exit.

RDIO: Read next sector of intermediate input. Increment sector address by one, reset address where next prefix word is obtained, and exit.

NOTE: It is possible for the Symbol Table to extend into the area where INT1 and INT2 reside if the assembly is in two pass mode with no listing.

Phase 12

Charts: CR, CS

- Processes END statements.
- Reads in Phase 10 to build a Program Header Record (Pass 1).
- Reads in Phase 11 (one pass mode) (Pass 1).
- Reads in and transfers control to Phase 3 (Pass 2).

ENDA: The branch table starts END processing here. If Pass 2, go to END2. Initialize switches for Pass 2, and load XR2 with BUFI plus 2, or DFBUF plus 2 if two or one pass mode, respectively.

CLR51: Clear first 51 words in DSF buffer to zero (maximum Program Header Record length). If relocation mode of assembly is relocatable (RLMDE), go to S2402. Otherwise, the program is type 1, and the length of the header (word 6 of header) is equal to three.

S2402: If ISS assembly, move the ISSNO (saved from Phase 2) to word 14 of the header, and the number of interrupt levels required to word 15. Move the number(s) of the interrupt levels specified with control records into the header beginning with word 16. Increment the length of the header by one for each level specified (ISHDR).

ISSXX: Reset XR2 to its value before moving the ISS information into the header. Set HDLTH equal to ISHDR and go to S2420.

S2403: If no ENTs used, go to S2404; otherwise, the program type is set to 4.

S2404: If ILS assembly, the interrupt level (saved in ISSNO) is inserted in word 13 of the Program Header, the Header length is 4, and the program type is 7.

S2420: If no EPR statement, go to S2421; otherwise, store extended precision indication in word 3 of Program Header Record, and go to S2430.

S2421: If no SPR statement, go to S2430, otherwise store standard precision indication in word 3 of the Program Header Record.

S2430: If program is type one, go to S2431. If program type is greater than two, go to S243A. If a file was not defined (no *FILE record), go to S2431. Store number of files defined (one) in word 9 of the Program Header Record. A seven word table (see S2485 in listing) is outputted to DSF by the DFOUT routine. As each word is outputted, the Location Assignment Counter is incremented by one. Thus the first data word will be assigned to relocatable seven in Pass 2. This table includes the size of the file (FIISZ) as obtained from the *FILE record.

S243A: If LIBR assembly (ISS or ENT called with one word call), reduce program type by one (type equals 5 or 3, respectively).

S2431: OR program type to precision already in word three of the Program Header Record.

S2440: Read Phase 10 from disk replacing the STADD section of Phase 9. Store the length of the header minus nine (HDLTH) in word 6 of the Header. Initialize the total word count of the program (WRDCT) to the length of the Header. Store the length of COMMON (completed in Phase 1) in word 5 of the Header. Set the data header address (DHPTR) equal to the address of the first word past the Program Header Record, and setup the XR load addresses in DFOUT to correspond. Note that these addresses will at this time be for the correct output buffer since XR2 was initialized for the correct buffer earlier in this phase.

S2151: If two pass assembly, go to ENDA2; otherwise, set word count equal to 320 at DFBUF, and sector address 32 relative to Working Storage at DFBUF plus one. Save the END statement in intermediate I/O (INT1), and force one more sector of intermediate output to be written, thus insuring that the END statement will be found in Pass 2.

ENDAO: The force-sector-write returns from INT1 to this address. Read Phase 11 (INT2) from disk, replacing the INT1 section of Phase 9. Force the reading of the first intermediate input sector.

ENDA1: The forced-read returns from INT2 to this address. Use INT2 to get the first source statement into the I/O buffer (SAREA). Clear the ENT and ISS switches for Pass 2 processing, and set-up FLIPR to read Phase 1 from the disk. Write the sector buffer area of the Symbol Table (possible partial sector) to the next available table overflow sector.

ENDXX: Address filled in here points to the sector overflow section of the Symbol Table.

ENDA2: Two pass assembly. Therefore it is necessary to reverse these addresses in the WRDFO subroutine to point to BUFI. Decrement the paper tape first position address (PTADR) by twenty so that the next record, if from paper tape, will read into position 1 of the I/O buffer. Use RDCRD to read the next record from the I/O device, and go to ENDA1 plus 2 to complete Pass 1 processing of END.

END2: Read in Phase 12A. Phase 12A performs END statement processing in Pass 2 beginning at P12AX.

Phase 12A

Chart: CT

P12AX: Adjust the Location Assignment Counter to the next even address if it is odd. Store the Location Assignment Counter in COM60 of the Skeleton Supervisor. Use DTHDR to complete the word count for the last data header, and then force the word count of this new data header to be zero. If program has an entry point, go to SBRTN; otherwise, program must have an execution address specified, or an S (syntax error) is inserted in the I/O buffer. The SCAN routine is used to compute the execution address which must be relocatable in a relocatable assembly. If it is not, ENDER is used to insert an R (relocation error) in the I/O buffer.

XEQOK: Store the execution address in COM56, and output in hexadecimal to positions 9-12 of the I/O buffer.

SBRTN: END statements with no execution address required or with an error pertaining to the execution address transfer to this address. The LDLBL routine is set up to bypass the read of the next record and to do only output options (print and/or punch).

ENDA5: The special entry to LDLBL returns to this address. If the DSF output is not past the three hundred and twentieth word of the buffer, go to ENDA3. Otherwise, use WRDFO to write output sector.

ENDA3: Use WRDFO to write last sector of DSF output.

ENDA4: Set up FLIPR to read Phase 3 from disk. Compute the block count of the program by dividing the total word count (WRDCT) by 20. Store the disk block count in COM54 of the Skeleton Supervisor and overlay with Phase 3.

ASSEMBLER INPUT-OUTPUT ROUTINES

Input/output routines for the Assembler include:

1. DISK0 – This routine is part of the Skeleton Supervisor.
2. CARD0/PAPT1 – This routine services either card or paper tape depending on which is defined as the principal I/O device for the particular system.
3. WRTY0/VIPST – This routine services either the Console Printer or the 1132 Printer, depending on which is defined as the principal printer for the particular system.

Input-Output Device Routine

The Assembler is device-independent; i.e., it is indifferent to the I/O device which it uses. The assembler uses whichever I/O device routine, card or paper tape, which was loaded by the System Loader/Editor.

The I/O device routine reads the input record and converts it into unpacked EBCDIC code in the input-output buffer SAREA. No return is made to the Assembler until the reading and conversion of the entire record is complete.

After completion of the Assembler control record input, the Assembler, by examining word 97_{10} in COMMA, determines if the input is from paper tape. If so, the Assembler shifts the pointer to the input buffer 20 character positions to the right.

As the assembly of each statement is completed, the Assembler builds the output record in the buffer SAREA. Card punching occurs from positions one through twenty of this buffer. The card output routine performs the character conversion. No return is made to the Assembler until the entire card has been punched.

If the output is to paper tape, the output record is built in the buffer SAREA and then moved to the buffer PBUF. Paper tape punching occurs from this buffer beginning at position one, ending at the end-of-record. The paper tape output routine performs the character conversion. As soon as punching has been initiated, the paper tape output routine returns to the Assembler, thus allowing for overlap of the paper tape output operation.

Print Device Routine

As with the principal I/O device, the Assembler is indifferent to the device assigned as the principal print device; i.e., the Console Printer or the 1132 Printer. The Assembler uses whichever routine was loaded by the System Loader Editor.

Upon entry, the move routine (P9MVE) determines if any I/O operation is in progress. If so, the move routine waits for the completion of that operation. Then the contents of the output buffer SAREA are moved to the print buffer PAREA. (This is the same buffer used for paper tape output, PBUF, renamed.) When the move is completed,the print line is initiated and the print routine returns to the Assembler.

The Assembler will not call for the new page routine (Phase 5) if the principal print device is the Console Printer.

The Console Printer output has as its first character a new line character to restore the type ball before starting each line. The 1132 Printer routine is set up to space after printing each line and to double space after printing at channel 1. The Printer routine has an entry labeled RPAGE to eject to a new page. A corresponding address in the Console Printer routine is a dummy entry to make the Assembler more independent with regard to the print device.

Overlays

To conserve core, the Assembler Program is divided into 15 phases (numbered 0-12A). Only Phase 9 and a portion of Phase 0 remain in core during the entire assembly process.

Statement processing phases (5, 6, 7, 8, and 12) are read in whenever they are needed. A branch table (Figure 3) is included at the beginning of each statement processing phase. When the statement type is determined (Phase 9), a branch is executed to the instruction in the branch table corresponding to the statement type. If the phase needed to process the statement type is in core, processing begins. If the phase is not in core, the branch table returns control to Phase 9. Phase 9 initializes the routine used to read disk data (FLIPR) and the requested phase is read into core and the program returns to the branch table to process the statement.

Statement Type	Phase Needed	Phase Needed is in Core	Phase Needed is not in Core
		BSC L to	BSC L to
DC	6	DCA	GETS6
INST	6	INSTA	GETS6
EQU	5	EQUA	GETS5
ORG	5	ORGA	GETS5
BSS,BES	5	BSSA	GETS5
EBC	8	EBCA	GETS8
CALL,LIBF	8	CALLA	GETS8
EXIT	8	EXITA	GETS8
LINK	8	LINKA	GETS8
DSA	8	DSAA	GETS8
DEC,XFLC	7	DECA	GETS7
END	12	ENDA	GTS12
HDNG*	5	HDNG5+2	GETS5

*When HDNG occurs while in Phase 2, reads in Phase 5 and branches to HDNG5. When HDNG processing is completed, Phase 2 is restored and control is returned to Phase 2 at HDRT2. In all phases after Phase 2, one of the two options indicated in the table is executed.

SECTION 5: FORTRAN

This section describes the internal structure of the 1130 Disk Monitor System FORTRAN Compiler programs which are designed to compile FORTRAN source statements into object programs. The source statements must conform to the statement specifications given in the 1130 FORTRAN Language publication (Form C26-5933).

PROGRAM PURPOSE

The FORTRAN Compiler accepts statements of a source program written in the 1130 FORTRAN language as input (see Figure 12) and translates them into machine language instructions, which form the object program. The object program can then be loaded, along with the required subroutines, for execution.

The compiler-generated machine language coding includes a large percentage of branch instructions which transfer control to subroutines during execution of the program. Thus, it is the subroutines that perform the majority of the operations in any given problem; however, it is the compiler-generated coding that selects and directs the subroutines.

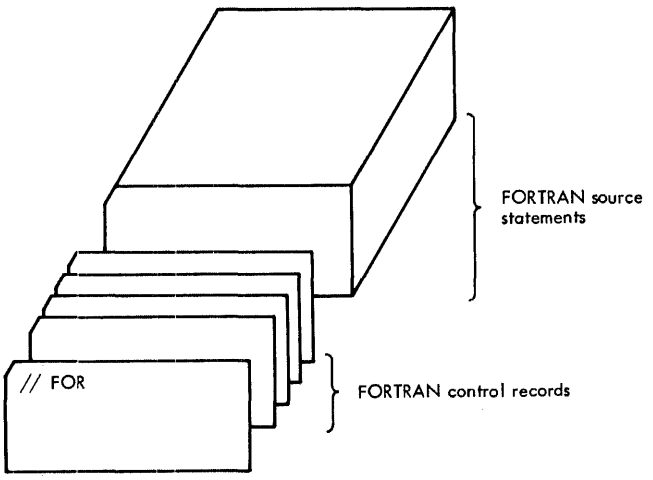


Figure 12. FORTRAN Input (card form)

GENERAL COMPILER DESCRIPTION

The FORTRAN Compiler consists of a series of 28 phases. Generally grouping the phases, Phases 1 and 2 are initialization phases which set up the compiler-reserved areas, interrupt levels, etc. Phases 3, 4, and 5 are the control phases which read control records, set Communications Area indicators, read the source statements, and ready them for the compilation phases. Phases 6 through 11 are specification phases which process the specification statements and other definitive information. Phases 12 through 20 are the analysis phases which perform the actual compilation. Phases 21 through 27 are the output phases, and Phase 28 is a recovery phase which restores the Skeleton Supervisor.

PHASE OBJECTIVES

The following is a list of the compiler phases and their major objectives.

1. First Sector — initializes the loader routines.
2. Second Sector — stores the Skeleton Supervisor on the disk and loads the Input Phase.
3. Input — reads the control records and source statements.
4. Classifier — determines the statement type and places the type code in the ID word.
5. Check Order/Statement Number — checks for the presence and sequence of SUBROUTINE, FUNCTION, Type, DIMENSION, COMMON, and EQUIVALENCE statements and places statement numbers in the Symbol Table.
6. COMMON/SUBROUTINE or FUNCTION — places variable names and dimension information in the Symbol Table; checks for a SUBROUTINE or FUNCTION statement and, if found, places the name and dummy argument names in the Symbol Table.
7. DIMENSION/REAL and INTEGER — places DIMENSION statement information in the Symbol Table and indicates the proper mode for REAL and INTEGER statement variables.
8. Real Constant — places the names of real constants in the Symbol Table.
9. DEFINE FILE — checks the syntax of DEFINE FILE, CALL LINK, and CALL EXIT statements and determines the defined file specifications.
10. Variable and Statement Function — places the names of variables, integer constants, and statement function parameters in the Symbol Table.
11. FORMAT — converts FORMAT statements into a special form for use by the object time

input/output routines.

12. Subscript Decomposition — calculates the constants to be used in subscript calculation at object time.
13. Ascan I — checks the syntax of all arithmetic, IF, CALL, and statement function statements.
14. Ascan II — checks the syntax of all READ, WRITE, FIND and GO TO statements.
15. DO, CONTINUE, etc. — replaces DO statements with a loop initialization statement and inserts a DO test statement following the DO loop termination statement. This phase also handles STOP, PAUSE, and END statements.
16. Subscript Optimize — replaces subscript expressions with an index register tag.
17. Scan — changes all READ, WRITE, GO TO, CALL, IF, arithmetic, and statement function statements into modified Polish notation.
18. Expander I — replaces READ, WRITE, GO TO, and RETURN statements with object coding.
19. Expander II — replaces all CALL, IF, arithmetic, and statement function statements with object coding.
20. Data Allocation — allocates an object time storage area for all variables.
21. Compilation Errors — prints out unreferenced statement numbers, undefined variables, and error codes for erroneous statements.
22. Statement Allocation — determines the storage allocation for object program coding.
23. List Statement Allocation — lists the relative statement number addresses, if requested.
24. List Symbol Table — lists the subprogram names from the Symbol Table and System Subroutine names found in the statement string, if requested.
25. List Constants — computes the addresses of the constants and lists them, if requested.
26. Output I — builds the program and data header records and places them into Working Storage. This phase also outputs the real and integer constants into Working Storage.
27. Output II — completes the conversion of the statement string to object coding and places the object program into Working Storage.
28. Recovery — restores the Skeleton Supervisor and returns control to it.
29. Dump — dumps the contents of the Symbol Table, Communications Area, and String Area, upon request.

CONTROL RECORDS

The FORTRAN control records are discussed in detail in the publication IBM 1130 Card/Paper Tape Programming System Operator's Guide (Form C26-3629).

CORE STORAGE LAYOUT

Figure 13 illustrates the layout of core storage during compilation. The hexadecimal addresses listed are approximate and should not be construed as the final address assignments. For the actual addresses, refer to the program listings using the symbolic labels.

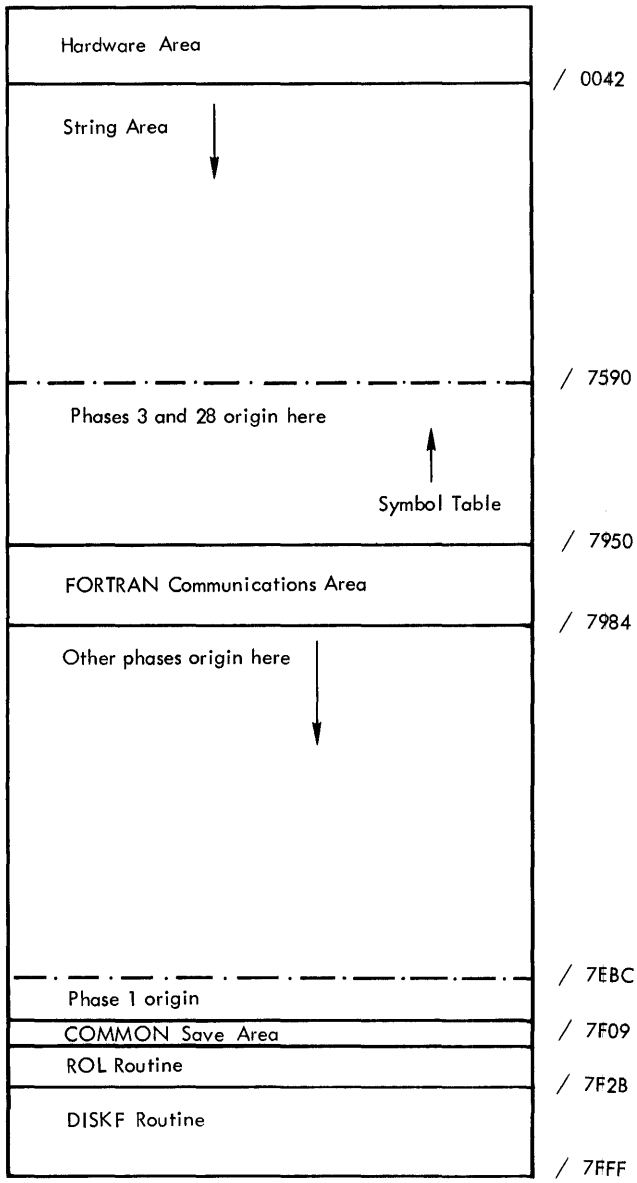


Figure 13. Layout of Storage during Compilation

Compiler-reserved Areas

Two areas of storage are reserved by the compiler. The first area, in the low-addressed words of storage, is comprised of approximately 66 words of storage. In this area are the index registers, the Interrupt Transfer Vector (ITV), interrupt traps, information pertaining to the restoration of the Skeleton Supervisor, and a CALL EXIT routine.

The second area, in the high-addressed words of storage, is comprised of approximately 247 words of storage. In this area are stored the ROL, DISKF, and DUMP routines. The DUMP routine calls the Dump Phase upon request; the ROL and DISKF routines control and execute the phase-to-phase transfer.

Phase 1, of which these routines are a part, is loaded into these locations by the Supervisor when the FORTRAN Compiler is initialized. These three routines are resident in storage all during the compilation.

FORTRAN Communications Area

The FORTRAN Communications Area consists of 52 words of storage where information obtained from the control records and compiler-generated addresses and indicators are kept. This information is available to any phase needing it. The contents of the FORTRAN Communications Area words are described in Table 5.

Phase Area

The Phase Area is the area into which the various phases of the compiler are read by the ROL routine. The size of the Phase Area is determined by the size of the largest phase of the compiler.

Each phase, when loaded into the Phase Area, overlays the preceding phase. There are three phases, however, which are exceptional in that they are loaded at some location other than the Phase Area origin. Phase 1 is loaded into high-addressed storage so that the ROL and DISKF routines occupy initially the positions they will occupy throughout the compilation. The control card analysis portion of Phase 3 is loaded into the String Area. This portion of the phase is in use only until the control cards have been processed and is overlaid by the source statements. Phase 28, the Recovery phase which follows the compilation, is also loaded into the String Area. The compilation, however, is completed by the time this phase is loaded. If the Recovery Phase should be loaded during compilation, due to an overlap or disk error, the Statement String is overlaid by the Recovery Phase.

COMMON Save Area

The COMMON Save Area is composed of nine words of storage used to save core location ten and various addresses which are used by the Recovery Phase, if and when it should be called.

Symbol Table

The Symbol Table contains entries for variables, constants, statement numbers, various compiler-generated labels and compiler-generated temporary storage locations.

The first entry of the Symbol Table occupies the three highest-addressed words of the String Area; i.e., the 3 words just below the first word of the Communications Area. The second entry is positioned in the lower-numbered core storage words adjacent to the first entry, etc.

During the initialization of the Symbol Table in Phase 3, three words of storage are reserved for the first Symbol Table entry. This entry is not made, however, until Phase 5. From this point the size of the Symbol Table varies from phase to phase until it achieves its largest size in Phase 19. Its size always includes the three words reserved for the next Symbol Table entry.

During Phases 5 through 19, the Symbol Table contains variables, constants, and statement numbers. Information for these entries has been removed from the statement string and has been replaced by pointers to corresponding Symbol Table locations. Also, the Symbol Table contains the various compiler-generated labels and temporary storage locations used in compilation.

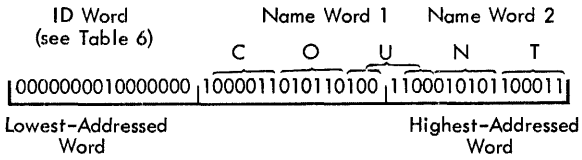
During the output phases, 20 through 27, these entries in the Symbol Table are replaced by object time addresses which are inserted into the object coding by Phase 27.

Table 5. FORTRAN Communications Area

Word	Symbolic Name	Description of Contents
1	SOFS	The address of the start of the string.
2	EOFS	The address of the end of the string.
3	SOFST	The address of the start of the Symbol Table.
4	SOFNS	The address of the start of the non-statement-number entries in the Symbol Table.
5	SOFXT	Phases 1-20. The address of the start of the Symbol Table entries for SGTs (subscript-generated temporary variables). Phases 21-25. The work area word count.
6	SOFGT	Phases 1-20. The address of the start of the Symbol Table entries for GTs (generated temporary storage locations). Phases 21-25. The constant area word count.
7	EOFST	The address of the end of the Symbol Table.
8	COMON	Phases 1-19. The address of the next available word for COMMON storage. Phase 20. The address of the highest addressed word reserved for COMMON storage. Phase 21 – Not used. Phases 22-25 – Relative entry point.
9	CSIZE	All phases except Phase 20. The COMMON area word count. Phase 20. The address of the lowest-addressed word reserved for COMMON storage.
10	ERROR	Bit 15 set to 1 indicates overlap error. Bit 14 set to 1 indicates other error.
11-12	FNAME	The program name (obtained from the *NAME control record, or SUBROUTINE, or FUNCTION statement). Stored in name code.
13	SORF	Set positive to indicate FUNCTION. Set negative to indicate SUBROUTINE.
14	CCWD	Control card word. Bit 15 set to 1 indicates Transfer Trace. Bit 14 set to 1 indicates Arithmetic Trace. Bit 13 set to 1 indicates Extended Precision. Bit 12 set to 1 indicates List Symbol Table. Bit 11 set to 1 indicates List Subprogram Name. Bit 10 set to 1 indicates List Source Program. Bit 9 set to 1 indicates One Word integers. Bit 8 set to 1 indicates Save Loader. Bit 6 set to 1 indicates Console Printer as the principal print device. Bit 5 set to 1 indicates 1132 Printer as the principal print device.
15	IOCS	IOCS Control Card Word. Bit 15 set to 1 indicates 1442 Card Read Punch. Bit 14 set to 1 indicates Paper Tape. Bit 13 set to 1 indicates Console Printer. Bit 10 set to 1 indicates Keyboard. Bit 9 set to 1 indicates 1132 Printer. Bit 8 set to 1 indicates Disk.
16	DFCNT	Define File Count.

Following the above 16 words are 36 words containing the IOCC words. The contents of these words is described in the compiler listings.

An entry for a subprogram name of COUNT would appear as:



Entry in hexadecimal form – 0080 86B4 C563

ID Word. The layout of the Symbol Table ID word is given in Table 6. The ID word is formed when the entry is placed in the Symbol Table.

Name-Data Words. Names in the Symbol Table are in a format similar to name code. However, the 30 bits comprising the name are equally divided between the two words. Bit zero of each word is set to zero for constants; it is set to one for all other Symbol Table entries.

Table 6. Symbol Table ID Word

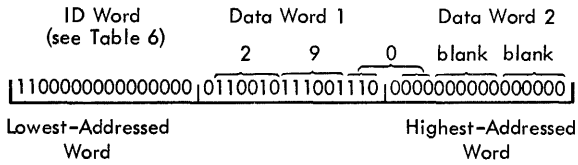
Bit Position	Status and Meaning
0	1 - Constant 0 - Variable
1	1 - Integer 0 - Real
2	1 - COMMON
3-4	01 - One dimension 10 - Two dimensions 11 - Three dimensions
5	1 - Dummy argument
6	1 - Statement number
7	1 - Statement function name
8	1 - Subprogram name
9	1 - FORMAT statement number
10	1 - Referenced statement number or defined variable
11	1 - External
12	1 - Generated temporary storage location (GT)
13	1 - Subscript-generated temporary variable (SGT)
14	1 - Allocated variable
15	Not Used

Format. All entries in the Symbol Table are comprised of three words; an ID word and two Name-Data words. The entries for dimensioned variables are exceptional, however, in that they are six word entries; the additional three words contain the dimension information.

The ID word occupies the lowest-addressed word of the three word entry. The Name-Data words occupy the two higher-addressed words.

In the six word, dimensioned variable entries in the Symbol Table, the ID word and Name-Data words occupy the same positions relative to each other. The dimension information is added in the three lower-addressed words below the ID word.

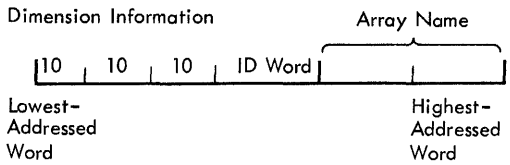
A typical entry is illustrated in the following example of an integer constant entry of 290.



Entry in hexadecimal form –E000 65CE 0000

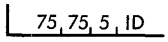
Dimensioned Entries. The Symbol Table entry for dimensioned variables requires six words: two for the array name, one for the ID word, and three for the dimension information. Three words are always used for the dimension information regardless of the number of dimensions.

For one-dimension arrays, the dimension words all contain the same information — the integer constant that specifies the dimension of the array. For example, the entry for array ARRY (10) would appear as:

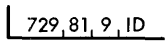


For two-dimensional arrays, one dimension word contains the integer constant of the first dimension and the remaining two words contain the product of the first and second dimension integer constants. For three-dimensional arrays, the

first two words are as they are for a two-dimensional array and the third word contains the product of the first, second, and third integer constants of the array dimensions. Thus, the dimension information for array B(5, 15) appears as:



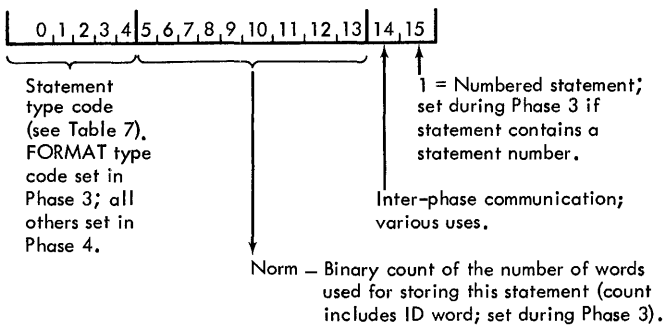
and the dimension information for array C(9, 9, 9) appears as:



Statement String

The source statements are read by the Input Phase, converted to EBCDIC, and stored in core storage. The first statement is stored just above Compiler-reserved area in lower storage and each succeeding statement is placed adjacent to the previous statement thus forming the source statements into a string. The area within which the source statements are stored is referred to as the string area.

ID Word. For identification purposes, as each statement is placed in the string area, an ID word is added at the low-address end of each statement:



The Norm is the only portion of the ID word completed by the Input Phase. The Norm is a count of the number of words used to store that statement, including the ID word and statement terminator.

The statement type codes, shown in Table 7 are added in the Classifier Phase.

Statement Body. Each statement, after being converted to EBCDIC, is packed two EBCDIC characters per word. This is the form in which the statements are initially added to the statement string.

Statement Terminator. Statements are separated by means of the statement terminator character, a semicolon. The statement terminator character indicates the end of the statement body. This character remains in the string entry throughout the compilation process. All statements in the

Table 7. Statement ID Word Type Codes

Code	Statement Types
00000	Arithmetic
00010	END
00100	SUBROUTINE
00110	CALL
00111	COMMON
01000	DIMENSION
01001	REAL
01010	INTEGER
01011	DO
01100	FORMAT
01101	FUNCTION
01110	GO TO
01111	IF
10000	RETURN
10001	WRITE
10010	READ
10011	PAUSE
10100	Error
10101	EQUIVALENCE
10110	CONTINUE
10111	STOP
11000	DO-test
11001	EXTERNAL
11010	Statement Function
11011	Internal Output Format
11100	CALL LINK, CALL EXIT
11101	FIND
11110	DEFINE FILE

statement string carry the statement terminator except for FORMAT and CONTINUE statements and compiler-generated Error entries. Error entries inserted into the string by the compiler are inserted without the terminator character.

String Area

The string area during compilation contains both the statement string and the Symbol Table. The statement string is built by the Input Phase beginning in the low-addressed words of the string area. The Symbol Table is built during the compilation process, beginning in the high-addressed words of the string area.

The sizes of the string and Symbol Table vary during the compilation. As some items are removed from the string, they are added to the Symbol Table.

In addition, some phases move the entire statement string as far as possible toward the Symbol Table. The last statement of the string then resides next to the last Symbol Table entry. As the phase operates on the statement string, now referred to as the input string, it is rebuilt in the low-addressed end of the string area. The rebuilt string is referred to as the output string. This procedure allows for expansion of the statement string.

If at any time during the compilation an entry cannot be made to the statement string or to the Symbol Table due to the lack of sufficient storage, an overlap error condition exists. In the event of such an overlap condition,the remaining compilation is bypassed and an error message is printed. (See the section, Compilation Errors.) Either the size of the source program or the number of symbols used must be decreased or the program must be compiled on a machine of larger storage capacity.

Compilation Errors

When an error is detected during the compilation process, the statement in error is replaced by an appropriate error entry in the statement string. Each entry is made during the phase in which it is detected and the procedure is the same in all phases.

1. The type code in the erroneous statement's ID word is changed to the Error type (see Table 3).
2. The statement body is replaced by the appropriate error number (see Table 7). The statement number, if present, is retained in the Symbol Table and the Symbol Table pointer is retained in the error entry on the string.
3. The statement string is closed up, effectively deleting the erroneous statement from the statement string.

Error entries in the Statement String are exceptional in that they do not carry the statement terminator character (semi-colon). CONTINUE and FORMAT statements also do not carry the special terminator character. All other statements in the string are ended by means of the semi-colon.

Error indications are printed at the conclusion of compilation. If a compilation error has occurred, the message

OUTPUT HAS BEEN SURPRESSED

is printed and no object program is punched.

Error messages appear in the following format:

CbAAbATbSTATEMENTbNUMBERbXXXXX YYY

where C indicates the FORTRAN Compiler, AA is the error number, XXXXX is the last encountered statement number, and YYY is the count of statements from the last statement number. See the publication IBM 1130 Monitor System Reference Manual (Form C26-3752) for a list of the FORTRAN error numbers, their explanations, and the phases during which they are detected.

In addition to the errors, undefined variables are listed by name at the end of compilation. Undefined variables inhibit output of the object program.

The initialization of each phase includes an overlap error check. If, at any time during the compilation, the statement string overlaps the Symbol Table, or vice-versa, the remainder of the compilation is bypassed and the message

PROGRAM LENGTH EXCEEDS CAPACITY

is printed.

PHASE DESCRIPTIONS

The description of the compiler operation is divided into separate descriptions for each phase. Each phase description is accompanied by a flow chart illustrating the logic flow of that phase. The symbols used on the charts are the same as those contained in the program listings.

Phase 1: First Sector

Chart: DA

- Loads Phase 2.

Upon detecting the // FOR Monitor control record, the Skeleton Supervisor loads the first sector of the FORTRAN Compiler. Phase 1 comprises this first sector.

The purpose of the phase is simply to load Phase 2 which performs the initialization needed to begin the compilation of the FORTRAN source program.

The phase is loaded into high-addressed storage rather than at the normal Phase Area origin. This is done in order to locate initially the ROL, DISKF, and DUMP routines into the compiler-reserved area in high-addressed storage. These routines reside in this area throughout the compilation and are used to control and execute the phase-to-phase transfer (ROL and DISKF) and to print the contents of the Symbol Table and statement string between phases (DUMP).

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or functions of the major routines and subroutines contained in Phase 1.

Routine/ Subroutine	Function
PHO	Initializes the disk interrupt; sets up the parameters for the disk read.
ROL	Loads the next phase from the disk into the phase area in memory.
DUMP	Tests the Console Entry Switches (1130) or Data Entry Switches (1800); if dump requested, sets up for the next phase; calls Dump Phase.
DISKF	Sets up the disk function parameters; tests disk for not-busy state and defective cylinders; accomplishes the READ, WRITE, and READ CHECK.
PLACE	Controls the READ, WRITE, SEEK, and READ CHECK disk functions.
SKO1	Performs the SEEK operation to a specified cylinder on the disk.

Phase 2: Second Sector

Chart: DB

- Places the Supervisor into the Core Image Buffer on the Disk.
- Loads Phase 3.

Phase 2 stores the Skeleton Supervisor in the Core Image Buffer on the disk. The CALL EXIT routine, which is loaded with Phase 2 but not executed, is placed into the compiler-reserved area in low-addressed storage. The defective cylinder table is saved for use by the FORTRAN disk routines.

Phase 2 loads Phase 3 through the ROL routine.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 2.

Routine/ Subroutine	Function
PHOA	Sets up the parameters for placing the Skeleton Supervisor onto the disk.
CALL1	Writes the first of two portions of the Skeleton Supervisor onto the first sector of the CIB.
CALL2	Writes the second of two portions of the Skeleton Supervisor onto the second sector of the CIB; saves the defective cylinder table; places the CALL EXIT routine into the reserved area in lower storage; loads Phase 3.

Phase 3: Input

Chart: DC

- Computes the core storage size.
- Reads the control records; sets indicators in the FORTRAN Communications Area.

- Reads the source statements; stores them in the string area; precedes each statement with a partially completed ID word.

- Checks for a maximum of five continuation records per statement.

- Lists the source program, if requested.

Phase 3 is composed of two major segments; the first analyzes the control records and the second inputs the source statements. The control record analysis portion of Phase 3 is loaded into the string area, while the statement input portion is loaded at the normal Phase Area origin.

The control record analysis routines, therefore, remain in storage until the processing of the control records is completed. They are then overlaid by the source statements as the statement string is built by the statement input portion.

Phase 3 begins the compilation process by determining the storage capacity of the compiling machine. The FORTRAN Communications Area is set up with pertinent addresses, indicators, and the computed storage size. The I/O interrupt levels and indicators are initialized and the control records are read. I/O controls and compiler indicators drawn from the control records are placed in the FORTRAN Communications Area. The control records are converted to modified EBCDIC and listed.

The source statements are now read. If the control records include an *LIST SOURCE PROGRAM or *LIST ALL record, the source statements are listed as they are read. At this time comment statements are bypassed from further processing. During input, a check is made for continuation statements. A maximum of five continuation statements per statement is permissible.

The body of the statement is packed, two modified EBCDIC characters per word. Blanks between characters or words are removed from all statements except FORMAT statements.

Phase 3 calculates the number of words required to store each statement on the string, including an ID word and two words for the statement number, if one is present. This number is called the statement Norm.

The statement is now stored on the statement string. The Norm is placed in the ID word and bit position 15 of the ID word is set to 1 for all statements having statement numbers.

Statement numbers are compressed into two words. Bits 15 and 16 of the second word are not used; statement numbers of less than five digits are left-justified, leading zeros being removed. The 2-word statement number is inserted between the ID word for the statement and the statement body.

Phase 3 handles FORMAT and END statements in an exceptional manner. The statement type code is inserted into the ID word of all FORMAT statements. For all other statements this function is provided in Phase 4.

When the source statement input routines detect the END statement in the source program, a special 1-word END indicator is placed onto the statement string rather than the END statement.

If, during Phase 3, an input record having a / (slash) in column one is detected, control is transferred to the Recovery Phase and the compilation is discontinued.

Errors Detected. The errors detected by Phase 3 are: 1 and 2.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 3.

Routine/ Subroutine	Function
AXXO	Reads the control records and source statements.
AXX2	Calls CONVT to convert the input buffer to EBCDIC; checks if the source statements are to be listed.
AXX3	Lists the source statements, if requested.
AXX4	Performs a check for comments statements.
AXX4A	Performs a check for continuation statements.
AXX5	Checks if more than five continuation statements are encountered.
AXX6	Checks for blank records.
AXX7	Calls PUT to place the ID word on the string; if present, puts the statement number on the string.
AXX11	Packs the statement body into two EDCDIC characters per word; puts the packed word into the string area.

Routine/ Subroutine	Function
AXX15	Inserts the statement type code into the ID word for FORMAT statements.
CCDO	Checks control record; stores control information in Communications Area.
CARDS	Controls column reading into the input buffer.
CONVT	Converts input characters to EBCDIC; converts characters to be printed into print code.
CSL	Checks for valid statement numbers, if present; packs the number into two words.
CNM	Removes trailing blanks from statements; computes the statement length (Norm); inserts the Norm into the ID word.
DEE	Resets interrupts; transfers to the ROL routine to load the next phase.
ERST	Places an error message on the string in place of an erroneous statement.
GETCH	Checks for an END record.
GTWD	Obtains characters from the input record; converts them to modified EBCDIC form.
HOBO	Picks up non-blank columns from the input records.
INTER	Transfers control to the specific device routines on an interrupt.
NEXT	Places the END indicator into the string.
PHO	Computes core size; initializes the FORTRAN Communications Area; sets up the I/O interrupts.
PRINT	Performs the printing onto the system printer.
PUT	Places a word into the string.
PUTS	Generates the special END indicator.
RTNMN	Transfers to the Recovery Phase if a / (slash) is detected in column 1 of an input record.
W8	Obtains the source program name from the *NAME control record; stores the name in the FNAME word in the FORTRAN Communications Area.

Phase 4: Classifier

Chart: DD

- Determines the statement type for each statement; inserts the type code into the statement ID word.
- Places the terminal character at the end of each statement.
- Converts subprogram names longer than five characters to five-character names.
- Converts FORTRAN supplied subprogram names according to the specified precision.
- Generates the calls and parameters which will initialize I/O routines at object time, if the IOCS control indicators are present.

Initially, Phase 4 moves the entire statement string next to the Symbol Table. As each statement is processed, it is moved back to the lower end of the string area.

According to the indicators set in the IOCS word of the FORTRAN Communications Area by the previous phase, Phase 4 generates the required calls to FORTRAN I/O. If the Disk indicator is on, a 'LIBF SDFIO' followed by its parameter is inserted in front of the statement string. Then, Phase 4 inserts the 'LIBF SFIO' followed by its parameters. Next, the table of device servicing routines is built and inserted.

See FORTRAN I/O in the Subroutine Library section of this manual for a description of these calls and parameters, an explanation of their function at execution time, and examples of the calls and calling sequences.

Phase 4, beginning with the first statement of the string, then proceeds to check each statement in order to classify it into one of the 31 statement types. FORMAT statements, already having the type code, and compiler-generated error messages are not processed by this phase.

Each statement name is compared to a table of valid FORTRAN statement names. Each recognized statement name is removed from the string and the corresponding ID type code is inserted into the statement ID word. (See Table 7 for the statement

ID type codes.)

Arithmetic statements are detected by location of the equal sign followed by an operator other than a comma. Because of this method of detection, arithmetic and statement function statements both carry the arithmetic statement ID type until Phase 10 which distinguishes them.

Names within the statement body are converted into name code and stored. Names with only one or two characters are stored in one word.

Phase 4 converts all parentheses, commas, etc. into special operator codes. These operators are each stored in separate words. Also, arithmetic operators (+, -, /, *, **) are each stored in separate words, and a statement terminator character (a semicolon) is placed after each statement, except CONTINUE and FORMAT statements and compiler-generated error messages.

NOTE: The string words containing name or constant characters have a one bit in bit position 0. Bit position 0 of string words containing arithmetic operator characters has a zero bit.

The standard FORTRAN supplied subprogram names specified in the source program are changed, if necessary, to reflect the standard or extended precision option specified in the control records. Also, the six-character subprogram names of SLITET, OVERFL, and SSWTCH, which are allowed so as to be compatible with System/360 FORTRAN, are changed to SLITT, OVERF, and SSWTC, respectively.

The word FUNCTION appearing in a Type statement and the statement numbers of DO statements are isolated by the Classifier Phase. Isolation is accomplished by placing a one-word special operator (colon) just after the word or name to be isolated. This process aids later phases in detecting these words and numbers.

Errors Detected. The error detected by Phase 4 is: 4.

Routine Summary. The following descriptions summarize the purpose or function of the major routines or subroutines contained in Phase 4.

Routine/ Subroutine	Function
BACK	Moves the entire statement string next to the Symbol Table.
CLFIO	Determines I/O specifications; generates the *FIO call.
CQCT	Checks for the presence of IOCS indicators in the FORTRAN Communications Area; sets up the parameters of the *FIO LIBF to reflect the specified precision.
ENDD	Checks for the presence of an END statement.
FCNT	Isolates the word FUNCTION in the string entry.
FIO	Table of I/O routine calls.
FUNEX	Functional name exchange table.
GET1	Gets a character from the string.
GET2	Gets two characters from the input string.
GETID	Initializes for a statement type code table search; gets the statement type code; stores it in the ID word.
IDAHO	Checks for the name SSWTCH.
MAKE	Stores names in compressed EBCDIC, five characters per two words; stores operators in one word on the string.
MOVE	Moves statements from the input string to the output string.
MOVIE	Updates the string pointer (XR1) to move to the next statement.
PSDIO	Places a call to single-disk I/O routines onto the string, if applicable.
PTFIO	Places the *FIO call into the string.
PUT	Places the word in the accumulator into the string.
QZA1	Inserts the new Norm in the statement ID word.
START	Checks the FORTRAN Communications Area ERROR word for overlap condition.
TWONT	Table containing the first two characters of FORTRAN statement names and the address of the remaining name characters.
WAIT	Transfers to the ROL routine to load the next phase.
XXYZ	Closes the string by one word and adjusts the statement Norm.
XYZ	Checks for the name OVERFL.
XYZ1	Checks for the name SLITET.

Routine/ Subroutine	Function
ZAO	Initializes for a scan of the statement string.
ZA1	Checks for the special END indicator.
ZA1A	Places the END statement ID word on the string.
ZA1B	Checks for arithmetic statements.
ZA2	Checks the arithmetic statement for the various types of operators.
ZA6	Places the ID word into the string.
ZA7	Checks for a DO statement.
ZA11	Isolates statement numbers in DO statements.
ZA13	Places the statement terminator (;) at the end of the statement.

Phase 5: Check Order/Statement Number

Chart: DE, DF

- Checks subprogram and specification statements for the proper order; removes any statement numbers from these statements.
- Checks to ensure that statements following IF, GO TO, RETURN, and STOP statements have statement numbers.
- Removes CONTINUE statements that do not have statement numbers.
- Checks the statements for statement numbers; checks the Symbol Table for a previous entry of the same statement number.
- Places the statement number into the Symbol Table; places the Symbol Table address into the string entry.

Phase 5 makes two passes through the statement string. The first pass checks to ascertain that the subprogram and specification statements are in the following sequence:

SUBROUTINE or FUNCTION statement
Type statements (READ, INTEGER)
EXTERNAL statements
DIMENSION statements
COMMON statements
EQUIVALENCE statements

Statement numbers assigned to the statements listed above are removed. A second check is made to ascertain that these statements precede the first executable statement of the source program.

Any CONTINUE statements that do not have statement numbers are removed from the statement string. A check is made to ensure that statements following GO TO, IF, RETURN, and STOP statements have statement numbers.

The SORF word in the Communications Area is appropriately modified if a SUBROUTINE or FUNCTION statement is present.

The second pass of Phase 5 scans the statement string for statements with statement numbers. Each unique statement number is placed into the Symbol Table and the address of the Symbol Table entry is placed into the string entry where the statement number previously resided.

All statements having statement numbers previously added to the Symbol Table (duplicates of other statement numbers) are in error.

After each statement number has been placed into the Symbol Table and the Symbol Table address placed into the string, a check is made to determine if the Symbol Table has overlapped the statement string area. If an overlap has occurred, an error is indicated in the FORTRAN Communications Area by setting bit 15 of the ERROR word to 1; any further processing is bypassed by an immediate transfer to the ROL routine to load the next phase. All remaining phases are bypassed except Phase 21, which prints the overlap error message.

Errors Detected. The errors detected by Phase 5 are: 5, 6, and 9.

Routine Summary. The following descriptions summarize the purpose or function of the major routines or subroutines contained in Phase 5.

Routine/ Subroutine	Function
ABEL	Checks the statement for a statement number.
CKRL	Checks for REAL statements.
CLOSE CLOZE	Replaces the erroneous statement with an error message; closes up the string.
EFF	Checks for the presence of transfer statements.
ENDST	Checks for the END statement; checks for a statement number in statements other than END.

Routine/ Subroutine	Function
EOP	Branches to the ROL routine to load the next phase.
INIT	Initializes the phase; checks for a previous overlap condition.
LOOK	Scans the Symbol Table for a duplicate statement number.
MOVE	Updates the string pointer (XR1) to move to the next statement.
NEXT	Checks for statement numbers in statements following transfer statements.
PUTIN	Places the statement number into the Symbol Table; updates the FORTRAN Communications Area to reflect changes in the table's length; replaces the string area statement number with the Symbol Table address where it is now located.
RMOVE	Removes the statement number.
RMOV1	Removes the statement from the string.
START	Checks for a previous overlap error.
ST1	Checks for a FUNCTION statement.
SUBRT	Checks for a SUBROUTINE statement.
TAG3	Checks for COMMON statements.
TAG4	Checks for EQUIVALENCE statements.
TAG5	Checks for CONTINUE statements.
TENT	Checks for DEFINE FILE statements.
TENT1	Checks for INTEGER, EXTERNAL, and DIMENSION statements.

Phase 6: COMMON/SUBROUTINE or FUNCTION

Chart: DG, DH

- Places COMMON statement variables into the Symbol Table; includes dimension information, if present.
- Removes COMMON statements from the string.
- Checks for a SUBROUTINE or FUNCTION statement.
- Places the names and dummy arguments of the SUBROUTINE or FUNCTION statement into the Symbol Table; deletes the statement from the statement string.
- Checks REAL and INTEGER statements for the word FUNCTION.

Phase 6 is a two pass phase. The first pass processes COMMON statements; the second pass processes SUBROUTINE and FUNCTION statements, including FUNCTION found in REAL and INTEGER statements.

Pass 1 of Phase 6 examines all COMMON statements, checking all variable names for validity. Variable names are considered valid if the name:

1. begins with an alphabetic character,
2. contains no special characters, and
3. contains no more than five characters.

Unique variable names found in COMMON statements are placed into the Symbol Table. Duplicate variable names are in error.

When dimension information is present in a COMMON statement, the Symbol Table entry is expanded to six words, the dimension constants are changed to binary format, and this binary information is inserted into the Symbol Table entry. The Symbol Table ID word is updated to indicate the presence of the dimension information and the level of dimensioning.

See the section SYMBOL TABLE for the format of dimension and non-dimension entries in the Symbol Table.

Upon completion of storing the statement information, the COMMON statement is removed from the string.

The second pass of Phase 6 checks for a SUBROUTINE or FUNCTION statement among the specification statements. If either is found, the SORF word in the FORTRAN Communications Area is modified to indicate whichever is applicable. The subprogram name is checked for validity. If valid, the name is added to the Symbol Table and the address of the Symbol Table entry is placed into the FNAME word in the FORTRAN Communications Area. Following this, the subprogram parameters are checked and, if valid, they are added to the Symbol Table and the statement is removed from the string.

REAL and INTEGER statements are examined for the presence of the word FUNCTION. If the

FUNCTION specification is found, the REAL or INTEGER statement is processed in the same manner as a FUNCTION statement, except that the subprogram mode is specified explicitly by the statement type.

Errors Detected. The errors detected by Phase 6 are: 7, 8, 10, 11, 12, 13, 14, and 15.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 6.

Routine/ Subroutine	Function
CHK	Checks for a COMMON entry of a parameter name.
CLOSE	Closes the string after replacing the erroneous statement with an error message.
DD1	Updates the string pointer (XR1) to move to the next string word.
DD2	Checks the name for validity.
DD3	Checks for a comma operator.
DO2	Checks for a valid subprogram name.
DO3	Checks for a SUBROUTINE or FUNCTION statement.
DTB	Checks for a left parenthesis or a statement terminator following the subprogram name.
FLOP	Checks for a valid parameter name.
LLPQ	Removes an erroneous statement from the string.
MOVE	Replaces an erroneous statement with an error message; closes up the string.
MV	Updates the string pointer (XR1) to move to the next statement.
NEX	Places a dimension constant into the Symbol Table.
NEXP	Checks for a comma, a right parenthesis, and an overlap error; indicates the dimensioning level in the Symbol Table ID word.
PH	Initializes the phase; checks for a previous overlap error.
PIECE	Places the parameter name into the Symbol Table; sets the parameter and type indicators in the Symbol Table; checks for a Symbol Table overlap; checks for a comma or right parenthesis.
PLACE	Places a variable name in the Symbol Table; sets the COMMON and type indicators in the Symbol Table ID word; checks for a Symbol Table overlap.
PLACQ	Places the subprogram name into the Symbol Table; places the address of the Symbol Table entry into the FORTRAN Communications Area (FNAME).
PLAC1	Sets the subprogram and type indicators in the Symbol Table ID word of a subprogram name found in a SUBROUTINE, FUNCTION, REAL FUNCTION, or INTEGER FUNCTION statement; checks for a Symbol Table overlap.
PRTE	Converts the dimension constant to binary.
PTB	Checks for the statement terminator.
RMOVE, RMV	Removes a statement from the string; closes up the string.
SOS	Branches to the ROL routine to load the next phase.
START	Initializes the phase; checks the first statement to see if it is a *FIO LIBF.
STAR1	Checks the FORTRAN Communication Area (SORF) for a subprogram indication; if none, checks the first statement to see if it is REAL or INTEGER.
TCNT	Checks that the dimensioning does not exceed three levels.
TRY	Checks for the word FUNCTION in a REAL or INTEGER statement; if found, indicates the function in the FORTRAN Communications Area.
TST	Checks for END and COMMON statement.
ZARRO	Scans the Symbol Table for the parameter name.
ZOR	Scans the Symbol Table for a duplicate of the subprogram name.
ZORRO	Scans the Symbol Table for duplication of the variable name.

Phase 7: DIMENSION, REAL, INTEGER, and EXTERNAL

Chart: DJ, DK

- Analyzes DIMENSION statements; places dimension information into the Symbol Table.
- Removes DIMENSION statements from the statement string.
- Places variables and dimension information from REAL, INTEGER, and EXTERNAL statements into the Symbol Table.
- Indicates in the Symbol Table ID word the mode (Real or Integer).
- Check EXTERNAL statements for the names IFIX and FLOAT, which are not allowed.

The processing of Phase 7 is done in two passes. The first pass analyzes DIMENSION statements.

Each variable name found in a DIMENSION statement is first checked for validity. If the name is valid, the Symbol Table is searched for a duplicate. If no duplicate is found, the variable name, along with its dimensioning information, is added to the Symbol Table.

If a duplicate is found which has not yet been dimensioned, the dimensioning information from the variable name is added to the existing Symbol Table entry. If a duplicate is found which has already been dimensioned, the variable name is in error.

In a subprogram compilation, a comparison is made to ensure that no variable name duplicates the subprogram name.

As dimensioned variables are added to the Symbol Table, the DIMENSION statements are removed from the statement string.

See the section SYMBOL TABLE for the format of dimension and non-dimension entries in the Symbol Table.

The second pass of Phase 7 examines the REAL, INTEGER, and EXTERNAL statements found in the statement string. Each variable found in these types of statements is checked for validity. Valid variables are compared to the Symbol Table entries. Those variables duplicated in the Symbol Table as the result of prior COMMON or DIMENSION statements are in error. Those not equated to Symbol Table entries are added to the Symbol Table in the same manner as in the first pass of this phase.

EXTERNAL statements are scanned for the names IFIX and FLOAT. These subprogram names are erroneous.

Errors Detected. The errors detected by this phase are: 7, 8, 16, 17, 18, 19, 20, 21, and 22.

Routine Summary. The following descriptions summarize the purpose or function of the major routine and subroutines contained in Phase 7.

Routine/ Subroutine	Function
BEGIN	Checks for a Symbol Table overlap.
BOB	Removes the statement from the string after a statement terminator is located; closes the string up.
CHK	Determines if the dimensioned variable duplicates the subprogram name.
CLOSE	Closes up the string after replacing an erroneous statement with an error message.
CLQSE	Replaces the erroneous statement with an error message.
COZ	Checks for a right parenthesis; indicates the dimensioning level in the Symbol Table ID word.
DAP	Checks for END, REAL, INTEGER, and EXTERNAL statements.
FUN	Checks a variable name for a FUNCTION or SUBPROGRAM name equivalent or previous dimensioning; initializes error routines, if required.
JAP	Checks for the statement terminator.
LAP3	Initializes to scan the body of REAL, INTEGER, and EXTERNAL statements.
LAP5	Sets the REAL indicator of the Symbol Table ID word.
LORD	Checks for a Symbol Table overlap.
MA	Sets the INTEGER indicator of the Symbol Table ID word.
MIX	Indicates a DIMENSION statement; initializes to scan the body of the statement.
MLTN	Checks the subprogram name for prior COMMON or DIMENSION reference; sets the EXTERNAL indicator in the Symbol Table ID word; checks for the name IFIX.

Routine/ Subroutine	Function
MOTQS	Moves the pointer; checks the next word for comma or left parenthesis; checks for a subprogram.
MV	Moves the pointer to the next statement.
NEX	Checks that the dimension constant is not zero; puts it into the Symbol Table.
NEXP	Checks for a comma or a right parenthesis; checks to see that there are no more than 3 dimensional parameters present per variable name.
NEXT	Puts the dimension constant into the Symbol Table.
NEXTS	Checks the next word for a comma; checks to see that there are no more than 3 dimensional parameters present.
PADS	Checks the variable name for validity.
PHASE	Initializes the phase; checks for a previous overlap error.
PHIL	Moves to the word following a comma or left parenthesis; tests for a dimension constant; converts the constant to binary.
PLACE	Places the variable name into the Symbol Table.
PLACQ	Places the variable or subprogram name into the Symbol Table.
PREV	Checks for previous dimensioning of a variable; checks SORF word for the subprogram indication; determines if name is in COMMON.
RMOVE	Removes the statement from the string; closes up the string.
SIP	Checks for the statement terminator. Converts the dimension constant to binary.
SOS	Transfers to the ROL routine to load the next phase.
SUBN, SUBQ	Spreads the Symbol Table so that dimensional parameters can be added.
TARZ	Moves the pointer; checks for a left parenthesis.
TCNT	Checks to see that there are no more than 3 dimensional parameters present.
TEST	Checks for END and DIMENSION statements.
TICKK	Checks for the name FLOAT.
TICKQ	Indicates Real or Integer in the Symbol Table ID word of a variable.
YELP, YELP1	Checks for a Symbol Table overlap; checks the next word for a comma.
ZAR	Checks for valid names in REAL, INTEGER, and EXTERNAL statements.
ZOR, ZORRO	Searches the Symbol Table for a duplicate entry.

Phase 8: Real Constants

Chart: DL

- Scans all IF, CALL, and arithmetic statements for valid real constants.
- Converts real constants to standard or extended precision format, as specified.

Phase 8 examines the arithmetic, IF, and CALL statements found in the statement string, checking for valid real constants.

Each valid real constant is converted to binary in the precision indicated by the FORTRAN Communications Area indicators derived from the control records in Phase 3. The Symbol Table is checked for a previous entry of the constant. If a previous entry is found no new entry is made and the Symbol Table address of the constant is inserted into the statement string along with the constant operator in place of the constant. If no previous entry in the Symbol Table is found, the converted constant is added to the Symbol Table and the constant operator with the Symbol Table address of the constant replaces the constant in the statement string. The statement string is closed up following the alteration of the string.

Errors Detected. The following error is detected in this phase: 23.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 8.

Routine/ Subroutine	Function
CAP	Checks for END, arithmetic, IF, and CALL statements.
CLSUP	Closes up the string; adjusts the Norm of the closed statement.
GET	Collects the elements of the real constant to be converted to binary.
MOVE	Moves to the next statement on the string.
OUT	Transfers to the ROL routine to load the next phase.
RC	Checks for the statement terminator; tests switch 6, indicating the string needs to be closed up.
RCC	Initializes to scan the body of IF, CALL, and arithmetic statements.
RC1	Gets a character from the constant; checks it for zero or a digit.
RC2	Checks whether the present character is E, if E is allowable.
RC3	Checks whether an exponent sign is present and valid, if signed numbers are allowable.
RC10	Checks the constant for validity.
RC19	Checks the specified precision of the program; combines the extended constant, if Extended Precision.
RC20	Combines the Standard Precision constant.
RC21	Checks the Symbol Table for a previous entry of a constant; enters constants into the Symbol Table if not previously entered; checks for an overlap error.
RC22	Inserts the constant operator and the Symbol Table address into the statement string; moves the statement pointer; checks whether statement closure is required; calculates the number of words for closure.
START	Checks for a Symbol Table overlap.
SWIT	Checks whether the present character is a decimal point, if a decimal point is allowable.
Z	Moves the pointer to the next word of the statement body.
ZZ	Checks the real constant for validity.
Z1	Checks for a decimal point.
Z3	Checks for an operator.
Z33	Moves the pointer to the last operator preceding the constant; initializes to collect the real constant.

Phase 9: DEFINE FILE, CALL LINK, CALL EXIT

Chart: DM

- Checks the syntax of DEFINE FILE, CALL EXIT, and CALL LINK statements.
- Determines the defined file specifications.

Phase 9 checks the syntax of all DEFINE FILE, CALL LINK, and CALL EXIT statements. All variable names are checked for validity and are added to the Symbol Table. All valid constants are converted to binary and are added to the Symbol Table.

The SORF word in the FORTRAN Communications Area is examined to ensure that a DEFINE FILE statement does not appear in a subprogram.

This phase then computes the file definition specifications; that is, a DEFINE FILE table comprised of one entry for each unique file. Each entry consists of, in order, the file number, the number of records per file, the record length, the associated variable, a blank word for insertion of the sector address at load time, the number of records per sector, and the number of disk blocks per file. A count is kept in the DFCNT word in the FORTRAN Communications Area of the number of files defined.

Errors Detected. The errors detected by this phase are: 3, 70, 71, 72, and 73.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 9.

Routine/ Subroutine	Function
CE1	Modifies the CALL EXIT ID word

Routine/ Subroutine	Function
	to match the CALL LINK ID word.
CK1	Initializes to scan the statement body.
CL1	Checks the syntax of the CALL LINK statement.
COLL	Converts a constant to binary.
DF1	Checks for file definition in a subprogram.
DF3	Places a variable name into the Symbol Table, if not previously entered.
DF5	Collects valid constants to binary; gets the file count, file number, number of records, and record length; checks variable names for validity.
ED1	Transfers to the ROL routine to load the next phase.
FILES	Checks that there are no duplicate files and no more than 75 files defined.
IDSV1	Checks for DEFINE FILE, CALL EXIT, CALL LINK, and END statements.
OUT	Places the contents of the accumulator into the output string.
SKIP	Bypasses the remainder of the statement.
XR2R	Computes the number of records per sector and disk blocks per file; updates the file count.

Phase 10: Variable and Statement Functions

Chart: DN

- Places variables, internal statement numbers, and integer constants into the Symbol Table.
- Places parameters from statement function statements into the Symbol Table.
- Converts operators according to the scan-forcing table.
- Converts the left parenthesis of subscripts to a special dimension indicator.

Phase 10 checks the variable names found in the statement string for validity. Valid variables are added to the Symbol Table. A second check is made to ensure that all variable names conform to the implicit or explicit mode specifications (Real and Integer). Integer constants and internal statement numbers are also added to the Symbol Table, provided they are unique. However, names, integer constants, and internal statement numbers that are found in subscript expressions are not added to the Symbol Table until a later phase.

When adding names, constants, and statement numbers to the Symbol Table, Phase 10 replaces them in the string by pointers to their respective Symbol Table entries. The pointer replacing a constant, name, etc. is the address of the ID word of the Symbol Table entry for that constant, name, etc.

This phase also examines statement function statements.

The ID word of the statement function statement, until now identical to that of an arithmetic statement, is changed to the statement function type. Also, the parameters of statement functions are added to the statement function name, located in the Symbol Table. These entries in the Symbol Table are distinguished by their lack of a sign bit in the second word of the name.

During Phase 10,the left parenthesis on subscripts is changed to a special left parenthesis operator which indicates the order of the dimension that follows.

This phase also converts all operators except those in subscripted expressions from the 6-bit EBCDIC representation to a pointer value. This pointer value is derived from the scan-forcing table. The conversion is done in preparation for the Scan Phase, Phase 17, when an operation priority will be determined through these pointer values.

Errors Detected. The following errors are detected in this phase: 7, 24, 25, 26, and 43.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 10.

Routine/ Subroutine	Function
COOT	Converts the constant to binary.
CVO	Changes the ID word of statement function statements from arithmetic type to the statement function type; checks for a statement number; if one is found, changes the statement number in the Symbol Table to a

Routine/ Subroutine	Function
	compiler-generated label; removes the statement number from the string; inserts into the string a pointer to the generated label in the Symbol Table.
CV1	Initializes to scan the statement body; checks for END, *FIO, CONTINUE, FORMAT, EQUIVALENCE, Error, CALL LINK, CALL EXIT, DEFINE FILE, and statement function statements.
CV2	Checks for the presence of a GO TO statement; checks for a left parenthesis operator, indicating a computed GO TO.
CV3	Checks for the presence of a DO statement.
CV3A	Checks for the presence of an IF statement.
CV4	Checks for the presence of READ/WRITE statements.
CV5B	Checks for a semicolon operator.
CV5C	Converts the semicolon operator to scan-forcing code.
CV5D	Closes up the string by the specified number of words; adjusts the statement Norm.
CV6	Checks for a constant operator; if present, removes it.
CV8	Checks for a left parenthesis operator.
CV9	Checks for a right parenthesis operator.
CV10	Converts the operator with the aid of the QX1 table to scan-forcing code.
CV11	Checks for a variable name; if present, checks it for validity.
CV12	Replaces the erroneous statement with an error message.
CV13	Places statement function names into the statement function name table; checks for duplicate dummy parameters; checks for a maximum of 15 entries in the statement function name table.
CV14A	Checks the statement function name table for previous entry of the statement function name.
CV15	Removes the sign bit from the second word of the statement function name; adds the dummy arguments indicator to the Symbol Table ID word being built.
CV16	Checks for an integer name; adds the integer indicator to the Symbol Table ID word being built.
CV19	Converts the left parenthesis to the special left parenthesis operator indicating subscripts and dimension level.
CV20	Moves the pointer; checks for a semicolon operator or a right parenthesis operator.
CV22	Checks the constant for validity.
CV25A	Checks the statement number for validity.
CV26	Tests for a semicolon operator.
CV27	Checks for a right parenthesis operator.
CV31	Checks for an apostrophe (') mark separating the parameters of a disk READ or WRITE statement.
CV31A	Checks for the statement number of a FORMAT statement in a READ or WRITE statement.
DIMED	Checks for a dimensioned name and arithmetic statement.
EXTRA	Places the Symbol Table address of the statement function name into the statement string; closes up the string.
NASFT	Checks the statement function name table for a duplicate entry.
ORGIN	Initializes the phase; checks for a Symbol Table overlap.
PUTF	Checks for a Symbol Table overlap.
STOOK	Checks the Symbol Table for a previously entered statement function name; if not found, enters the name, along with the ID word constructed.
SXB1	Closes up the statement by 1 word.
TAM	Moves the pointer; checks for a left parenthesis operator.
WAIT	Transfers to the ROL routine to load the next phase.
YYX	Checks for a duplicate statement number in the Symbol Table.

Routine/ Subroutine	Function
YYZ	Checks for a DO statement.
YY2	Indicates in the Symbol Table ID word that a statement number is referenced.

Phase 11: Format

Chart: DP

- Converts FORMAT statements to a format for interpretation by the FORTRAN I/O routines.
- Converts the Apostrophe (') type format to H-type.

Phase 11 first moves the statement string into the upper end of the string area adjacent to the Symbol Table. The string is then scanned for FORMAT statements. All statements are moved to the output string at the lower end of the string area unchanged, except for FORMAT statements.

FORMAT statements are checked for a statement number. Those without statement numbers are in error. Numbered FORMAT statements are checked for valid types and syntax. All formats are checked for the Apostrophe (') type specification. Any of this type that are detected are changed to H formats.

In decomposing the FORMAT statement, Phase 11 converts each format into a format specification. (See Table 8.) Where required, the Field Repeat, Group Repeat, and REDO counts are computed and inserted. At the completion of Phase 11, the FORMAT statement is simply a chain of format specifications.

Table 8. Output Code of FORMAT Specifications

Type	Specification 4 Bits	5 Bits	7 Bits
E-	0 0 0 0	DD	WW
F-	0 0 0 1	DD	WW
I-	0 0 1 0		WW
A-	0 0 1 1		WW
X-	0 1 0 0		WW
H-	0 1 0 1		WW
	0 1 1 0	Not Used	
/	0 1 1 1	Undefined	
Group Repeat	1 0 0 0		NO RR_1
Field Repeat	1 0 0 1		NO
	1 0 1 0	Not Used	
REDO	1 0 1 1		RR

DD	decimal width has a maximum of 127 (used only in E and F type formats)
WW	total field width has a maximum of 127 in E and F type formats, 145 in I, A, X, and H type formats
RR_1 (Group Repeat)	negative count to the first specification of the group to be repeated
RR (REDO)	positive count to the specification following the first open (right) parentheses to the left
NO	positive count of repetitions to be made of a field or group

Errors Detected. The following errors are detected in this phase: 27, 28, 29, and 30.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 11.

Routine/ Subroutine	Function
ABEL	Checks for the END statement.
B	Indicates F-type format in the format specification.
BAKER	Replaces erroneous FORMAT statements by error messages.
BOY	Sets the carriage control character into the T-type format specification.
BTEST	Checks for a right parenthesis; puts the T-type format specification into the statement; inserts the Field Repeat word.
C	Indicates E-type format in the format specification.
CBEL	Detects FORMAT statements in the string.
D	Indicates I-type format in the format specification.
DECEM	Checks FORMAT statements for the presence of a statement number.
DOG	Calculates the field width; checks that the Field Width specification in I-, A-, X-, and H-type formats does not exceed 145.
ECHO	Checks for the / (slash); checks for the right parenthesis; checks for the T-type format.
F	Checks X- and H-type formats for repeat specifications; checks width specifications; outputs the Field Repeat word.
FOX	Generates and outputs the Group Repeat word.
GET	Gets a word from the FORMAT statement body for analysis.
GRP	Checks for the carriage control characters in T-type formats.
GRP1	Checks the value of the Group Repeat indicators.
HAN1	Places the format specification into the FORMAT statement.
I	Indicates A-type format in the format specification.
INDIC	Sets the indicator in the Symbol Table ID word to denote that the statement number refers to a FORMAT statement; initializes to scan the statement body; gets a word of the statement body; checks for the left parenthesis.
INK	Checks for invalid apostrophe (') marks.
K	Inserts the Decimal Width and Field Width into the format specification.
KILO	Gets the next word of the statement body; checks for apostrophe (') marks.
LOB1	Places a special apostrophe (') mark on the string; checks that the Field Width in I-, A-, X-, and H-type formats does not exceed 145.
LOB3	Checks for double apostrophe (') marks in the statement.
LXQ	Moves the statement string adjacent to the Symbol Table.
M	Checks to see that the Decimal Width in E- and F-type formats does not exceed 127 or that the Decimal Width is not greater than the Field Width.
MAN	Outputs a REDO Count if a Group Repeat is indicated.
NOVEM	Moves non-FORMAT statements to the output string.
O	Outputs H-type specification.
OUT	Transfers to the ROL routine to load the next phase.
RP	Generates a slash (/) into the output string when a format is to be repeated.
START	Checks for a Symbol Table overlap.
TST	Checks for specification types F, E, I, H, X, A, or a left parenthesis.
TST1	Checks that the Field Width in E- and F-type formats is not greater than 127.
ZX	Indicates X-type format in the format specification.

Phase 12: Subscript Decomposition

Chart: DQ

- Calculates the constants needed for object time subscript computation.
- Sets up dummy arguments for insertion of object time variables.

Phase 12 bypasses immediately all FORMAT, *FIO, CONTINUE, and Error statements. All other statements are scanned but only those statements that contain the special left parenthesis operator inserted by Phase 10 are operated upon.

The subscripting information for each variable is checked for validity.

Phase 12 then calculates the subscript constants D_4 and D_1, D_2, and D_3 depending on the dimensioning level. See below for the method of derivation of these constants.

These subscript constants are inserted into the subscript expression with the subscript indices. The right and left parentheses enclosing the subscript expression are then changed to special operators to be used in a later phase.

Calculation of the Subscript Constants. Assuming the maximum subscript form

$$C * I + C'$$

Phase 12 computes the subscript constants D-factors as follows:

For a 1-dimension array — $A(C_1 * I + C_2)$

$$D_1 = C_1 * S$$
$$D_4 = (C_2 - 1) * S$$

For a 2-dimension array — $A(C_1 * I + C_2, C_3 * J + C_4$

$$D_1 = C_1 * S$$
$$D_2 = L * C_3 * S$$
$$D_4 = \left[(C_2 - 1) + I * (C_4 - 1)\right] * S$$

For a 3-dimension array — $A(C_1 * I + C_2, C_3 * J + C_4, C_5 * K + C_6) C_5 * K + C_6)$

$$D_1 = C_1 * S$$
$$D_2 = L * C_3 * S$$
$$D_3 = L * M * C_5 * S$$
$$D_4 = \left[(C_2 - 1) + L * (C_4 - 1) + L * M * (C_6 - 1)\right] * S$$

In these formulas,

L = first dimension factor
M = second dimension factor
S = size in words of the array entries
S = 1 for 1 word integers
S = 2 for Standard precision
S = 3 for Extended precision

C_1 and C_2 = constants in the first dimension value

C_3 and C_4 = constants in the second dimension value

C_5 and C_6 = constants in the third dimension value

I, J, and K are the subscript indices.

Errors Detected. The following errors are detected in this phase: 31, 32, 33, 34, and 35.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutine contained in Phase 12.

Routine/ Subroutine	Function
ABEL	Checks for statement function statements with subscripting; if found, initializes the error routine.
ABELL	Gets the dimension information from the Symbol Table entry to decompose the subscript expression.
C	Complements the constant; stores it; initializes for the next segment of the scan.
CK1	Checks for a duplicate name in the Symbol Table.
CK3	Converts the name to a pointer to the Symbol Table.
CK6	Determines the level of dimensioning by means of the special left

Routine/ Subroutine	Function
	parenthesis inserted in Phase 10; calculates D_4 for 3 dimension variables.
CK8	Calculates D_4 for 1 dimension variables.
CK9	Calculates D_4 for 2 dimension variables.
CK11	Checks for agreement between the string dimension indicator and the extracted dimension factor; checks for an EQUIVALENCE statement with a dimensioned variable; checks for agreement between the dimension information and the number of subscripts.
CK12	Calculates the number of words needed to store the decomposed subscript on the string.
CK13	Closes up the statement.
CK14	Opens up the string.
COLL	Converts the dimension constant to binary.
E	Detects a non-dimensioned name; checks for the +, -, comma, and right parenthesis operators.
G	Determines if a word contains an operator, name, or constant; checks the constant for validity.
HOLD1	Checks the CCWD word for the Extended Precision indicator.
MOVE	Moves the string pointer to the next statement.
N	Replaces erroneous statements by error messages.
OUT	Transfers to the ROL routine to load the next phase.
START	Checks for a Symbol Table overlap; initializes the string pointer.
TAT	Places the special right hand parenthesis on the string.
TV	Checks the variable name for validity.
TEST	Checks for END, Error, FORMAT, *FIO, DEFINE FILE, CALL LINK, CALL EXIT, and CONTINUE statements; initializes to process all other types.
TEST1	Checks for the statement terminator; checks for 1, 2, or 3 dimension subscripts.
Z1	Puts a 1 dimension subscript onto the output string.
Z4	Puts a 2 dimension subscript onto the output string.
Z6	Puts a 3 dimension subscript onto the output string.
ZAPZ	Closes up the string.
ZAP6	Goes to the next statement in the string.
ZZZZ	Sets up the dimension indices; adds D_4 to the statement string.

Phase 13: A-Scan I

Chart: DR

- Checks the syntax of arithmetic, IF, CALL, and statement function statements.
- Checks statement function calls, including nested calls, for valid names and the correct number of arguments.
- Checks for the definition of variables; checks for valid statement number references in IF statements.

Phase 13 examines only the arithmetic, IF, CALL, and statement function statements in the statement string.

During the analysis of statement function statements a table is built containing the statement function name and the number of arguments associated with that function. This table is used in analyzing statement function calls, including nested calls, to check for the proper number of arguments.

The syntax of all arithmetic expressions is checked. When a variable is defined, i.e., found on the left side of an equal sign, the defined indicator is set in the ID word of the Symbol Table entry for that variable.

The statement number lists in IF statements are checked for proper syntax, for valid statement number references. References to FORMAT statements are in error.

The syntax of the record number expression in Disk READ/WRITE statements is checked. The right parenthesis is changed to a colon operator.

When detected in the following phase, the colon operator causes generation of a disk I/O operator rather than the standard I/O operator.

Errors Detected. The following errors are detected in this phase: 36, 37, 38, 39, 40, 41, 42, and 43.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 13.

Routine/ Subroutine	Function
ARITH	Scans all arithmetic expressions; checks statement function names for validity; generates a nested call table for statement functions; converts all operators to scan-forcing code.
CHECK	Checks for an overlap condition between the input and output statement strings.
CKASF	Ensures that all calls to statement functions have the correct number of arguments.
GETID	Gets the ID word of a Symbol Table entry.
LIST	Checks the statement number lists in IF statements for valid references.
MOVCH	Moves disk READ/WRITE statements to the output string; checks the syntax; converts the right parenthesis to a colon operator.
MOVE1	Moves one word from the input string to the output string; increments the pointer.
NAME	Checks names for dimensioning; determines the dimension level; moves the name and dimension information to the output string; allows space in the output string according to the dimension level.
ORGIN	Moves the statement string adjacent to the Symbol Table; initializes the input and output string pointers.
OUT	Moves words to the output string using index register 3 as a count control.
OVERF	Transfers to the ROL routine to load the next phase.
PUT	Moves the word in the accumulator to the output string; moves the pointer.
START	Checks for a Symbol Table overlap.
TESTV	Checks for valid variables in arithmetic expressions.
XY1	Initializes to scan the statement body.
XY2	Checks for the END statement; checks for a statement function table; if present, removes it.
XY5A	Checks for the equal sign; if found, initializes to scan the arithmetic expression.
XY5	Checks for an arithmetic statement; if found, checks for valid variable names; checks whether the name is defined.
XY7	Replaces the erroneous statement by an error message.
XY11	Moves a word to the output string; adjusts the statement Norm.
XY13	Checks for the statement terminator.
XY22	Moves the string pointer to scan the next statement.
XY23	Checks for an IF statement; if found, inserts the IF operator; checks for left and right parentheses; inserts the comma operator for the comma.
XY25	Detects CALL statements; checks for the subprogram indication.
XY26	Checks for disk READ/WRITE operators.
XY27	Moves a statement to the output string.
XY28	Checks for statement function statements; checks for a valid statement function name; moves the name into the statement function name table.
XY30A	Checks for a comma or right parenthesis.
XY30	Checks for a subscripted variable; if found, skips over it.
XY32	Checks for valid dummy arguments.
XY33	Counts the statement function arguments.

Phase 14: A-Scan II

Chart: DS

- Checks FIND, READ, WRITE, and GO TO statements for correct syntax, statement numbers and references, and variables.

- Detects implied DO-loops in READ and WRITE statements; generates the needed indicators.

The A-Scan II Phase examines all READ, WRITE, FIND, and GO TO statements.

When READ and WRITE statements are encountered in a mainline program, a check is made for the presence of IOCS indicators in the FORTRAN Communications Area. All READ and WRITE statements in a subroutine or function do not require the presence of IOCS indicators.

READ and WRITE Statements are checked for valid variables and FORMAT statement references and for proper syntax. Disk READ and WRITE statements are differentiated by means of the apostrophe ('). The appropriate I/O operators are generated for all READ and WRITE statements.

READ and WRITE statements are also checked for implied DO-loops. The necessary DO-initialize and DO-test operators are generated and inserted into the statement body.

GO TO statements are checked for proper syntax and the statement number lists are checked for valid statement number references. References to FORMAT statements are in error.

Testing the CCWD word in the FORTRAN Communications Area, Phase 14 determines if Tracing has been specified in the control records. If tracing, Arithmetic or Transfer, has been specified but IOCS has not, an error message is generated to this effect. However, if the SORF word in the FORTRAN Communications Area indicates that the program being compiled is a subprogram, no error message is generated.

Errors Detected. The following errors are detected in this phase: 43, 44, 45, 46, 47, 48, 49, 50, and 68.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 14.

Routine/ Subroutine	Function
CHECK	Checks for an overlap condition between the input and output statement strings.
CLMV1	Checks for the semi-colon operator in FIND statements.
DELET	Deletes one word from the input string.
DO	Checks for implied DO-loops in input and output lists; checks the syntax of the implied DO.
GETID	Gets the ID word from a Symbol Table entry.
LIST	Checks the statement number list in GO TO statements for valid references.
MOVE1	Moves one word from the input string to the output string; increments the pointer.
NAME	Checks names for dimensioning; determines the dimension level; moves the name and dimension information to the output string; allows space in the output string according to the dimension level.
ORGIN	Moves the statement string adjacent to the Symbol Table; initializes the input pointer; checks for the Trace and IOCS indicators in the FORTRAN Communications Area.
OUT	Moves words to the output string using index register 3 as a count control.
OVERF	Transfers to the ROL routine to load the next phase.
PUT	Moves the word in the accumulator to the output string; moves the pointer.
SKCOL	Checks for the colon operator; if found, changes the I/O operator to a disk I/O operator; changes the colon operator back to a right parenthesis operator.
START	Checks for a Symbol Table overlap.
TESTV	Checks for valid variables in READ, WRITE, and GO TO statements.
W11	Detects EQUIVALENCE statements.

Routine/ Subroutine	Function
XY1	Initializes to scan the statement body.
XY2	Checks for the END statement.
XY3	Detects READ statements.
XY4	Detects WRITE statements.
XY4B	Detects FIND statements.
XY5	Checks for the apostrophe (') mark; changes it to a comma operator.
XY6	Checks the SORF word for a subprogram indicator; checks for the presence of IOCS indicators in a mainline compilation.
XY6A	Checks the syntax of FIND, READ, and WRITE statements; generates I/O operators into the statement string; checks variable names and integer constants for validity; checks for a FORMAT statement number reference.
XY7	Replaces the erroneous statement by an error message.
XY8	Checks for the right parenthesis operator.
XY10	Checks for the semicolon operator.
XY11	Moves a word to the output string; adjusts the statement Norm.
XY14	Checks a variable for definition; if defined, sets the indicator in the Symbol Table ID word.
XY15	Checks for the range operator.
XY16	Checks for a non-dimensioned integer variable; indicates definition if defined; generates the DO-initialize operator.
XY24	Checks the syntax of GO TO statements; checks the statement number list for valid references.
XY27	Moves a statement to the output string.

Phase 15: DO, CONTINUE, STOP, PAUSE, and END

Chart: DT

- Checks the validity of DO statements and of nested DO-loops.
- Generates the coding needed to perform the DO-test.
- Checks the syntax of DO, CONTINUE, STOP, PAUSE, and END statements.
- Checks for a GO TO, IF, STOP, CALL LINK, CALL EXIT, or RETURN statement as the last executable statement of the source program.

Phase 15 examines DO, CONTINUE, CALL LINK, CALL EXIT, STOP, PAUSE, and END statements only.

Statements which follow STOP statements are checked to ensure that they are numbered statements. All integers found in PAUSE and STOP statements are checked to ensure that they are not greater than 9999. Valid integers are added to the Symbol Table as integer constants.

Detection of the END statement causes a check to be made of the last executable statement in the source program. An Error results if this statement is not an IF, GO TO, STOP, or RETURN statement.

While the DO statements are analyzed for correct syntax, Phase 15 constructs a DO table in the format given in Figure 14.

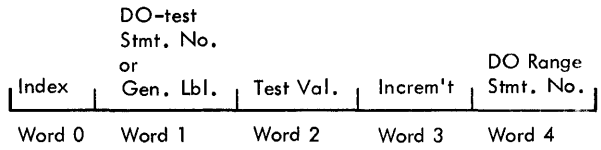


Figure 14. DO Table

A DO table entry is made for each DO statement when it is detected. As the statements following the DO statement are scanned, the statement numbers are compared with the contents of word 4, the range limit, of the DO table entries. When the range limit is found the DO-test coding is inserted into the statement string.

The DO table is built from low-to high-addressed storage. It is scanned, however, from high-to low-addressed storage. In this manner, nested DO loops which violate range limits are detected and an error message replaces the DO statement on the statement string.

The maximum number of entries in the DO

table is 25. Thus, no more than 25 nested DO loops can be contained in one program.

Phase 15 also checks to ensure that the last statement of the DO range is not a transfer statement.

Errors Detected. The following errors are detected in this phase: 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, and 62.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 15.

Routine/ Subroutine	Function
ABEL	Checks for END, DO, STOP, and PAUSE statements.
BAKER	Checks for numbered statements.
BOXX	Puts the pointer to a constant in the string; places the constant into the Symbol Table; checks for a Symbol Table overlap.
BTOD	Converts binary numbers to decimal.
CEE	Initializes to scan the statement; gets the statement number from the Symbol Table.
CHECK	Gets the ID word of a Symbol Table entry.
CLOSE	Replaces the erroneous statement with an error message; closes up the string.
DEMP	Places a zero as the increment into word three of the DO table entry; puts the test value constant into word two of the DO table entry.
DOTBL	Checks to ensure that the DO table is empty at the end of the phase.
FULL	Checks for more than 25 entries in the DO table.
HAH	Places the generated label or statement number for the DO-test into word 1 of the DO table entry.
HAP	Puts the end of range statement number into word 4 of the DO table entry; checks for valid variables; indicates definition on all defined variables.
HOHO	Checks for FORMAT statements.
LIZ	Checks for a DO-test statement; if not found, opens the string seven words; inserts the DO-test ID word.
MOVE	Moves the statement pointer to the next statement.
MOVE1	Changes the ID word of CONTINUE statements defining the DO range to the DO-test type; opens the string six more words.
MUIT	Checks for a numbered statement following the DO statement; if found, generates a label and inserts it in the Symbol Table if no statement number found.
NASTY	Inserts the special DO-test into the string when the increment is implied.
OUT	Transfers to the ROL routine to load the next phase.
OUT1	Checks for a transfer statement just prior to the END statement.
OVERL	Checks for a Symbol Table overlap; inserts the DO-test coding into the statement string.
PEND	Places the index variable into word 0 of the DO table entry.
PPEM	Checks the statement ID word for the "referenced" indicator; checks for a CONTINUE statement.
ROUT	Opens the string one word for insertion of a generated label.
SEM	Places the DO-initialize coding on the string; closes up the string.
START	Checks for a Symbol Table overlap; initializes the string pointer.
TED	Initializes to scan the statement; checks for the semicolon; opens the string one word; adjust the statement Norm; places the Symbol Table address of the constant on the string.
TINUE	Checks for an equivalent statement number in the DO table; checks for a transfer statement.
TRCKS	Checks the statement ID word for transfer statement types.
XYZ1	Places the DO ID word into the statement string.

Phase 16: Subscript Optimize

Chart: DU

- Scans READ, WRITE, IF, CALL, and arithmetic statements for subscript expressions.
- Optimizes subscript calculation by means of the subscript expression table.
- Inserts the SGT (subscript generated temporary).

This phase examines READ, WRITE, IF, CALL, and arithmetic statements. These statements are checked for subscript expressions.

Each unique subscript expression is placed into a table called the subscript expression table (see Figure 15). Each unique variable of a subscript expression is placed into a table called the bound variable table.

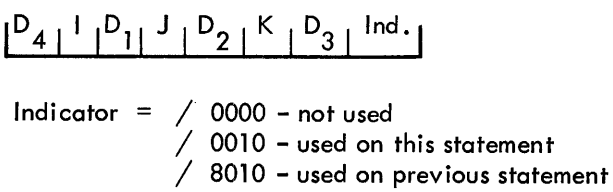


Figure 15. Subscript Expression Table

As each subscript expression is entered into the subscript expression table, it is removed from the string and replaced by a pointer to its location in the subscript expression table.

An SGT (subscript generated temporary) is generated for each entry made to the subscript expression table. The SGT is placed into an SGT table. The SGT is also placed into the Symbol Table and a pointer to the Symbol Table entry is inserted into the statement in the string.

For each subscript expression encountered, a scan is made of the bound variable table. If one or more of the variables in the subscript expression are not located in the bound variable table, the subscript must be re-calculated. Thus, a unique entry is made to the subscript expression table for this expression and an associated SGT is generated.

If, however, all the variables of the subscript expression are located in the bound variable table, the subscript expression table is then scanned to determine if a duplicate subscript expression is already located in the table. If no equivalent is found, the subscript expression is added to the table and a corresponding SGT is generated.

If a duplicate expression is found in the subscript expression table, the subscript expression is removed from the string and is replaced by a pointer to its duplicate in the subscript expression table. Identical subscript expressions now share the same indices of a common entry in the subscript expression table and the same SGT.

Whenever a variable is assigned a new value, i.e., appears to the left of the equal sign in an arithmetic expression, in the argument list of a subprogram, etc., that variable, if found in the bound variable table, is removed from the table. This removal from the bound variable table causes all entries in the subscript expression table containing that variable to be removed. The associated SGTs are also removed from the SGT table but remain in the Symbol Table. The string pointers to the SGTs in the Symbol Table also remain.

If a statement is encountered containing a subscript expression and having a statement number which is referenced by some other statement, the entire subscript expression table is cleared and all subscripts, beginning with the subscript expression of the referenced statement, must be re-calculated.

The subscript expression table is also cleared whenever a DO statement is encountered. Following subscripts must be re-calculated. Implied DO-loops, as in READ and WRITE statements, cause only those entries involving the index of the implied DO to be cleared. Only the subscripts involving that index must be re-calculated.

A maximum of 15 entries can be made into the subscript expression table. Thus, more than 15 subscript expressions in a single statement result in an error.

Errors Detected. The following error is detected in this phase: 63.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 16.

Routine/ Subroutine	Function
ANIML	Extracts a bound variable to be purged from the subscript expression table.
B	Checks for READ, WRITE, IF, CALL, and arithmetic statements.
B3	Checks for a non-dimensioned integer variable; if statement is arithmetic type, purges the variable from the subscript expression table.
B4	Purges a variable from the bound variable table.
CHECK	Gets the ID word of a Symbol Table entry.
D	Determines the dimension level of a subscript expression.
ECHO	Adds a subscript variable to the bound variable table.
F	Checks for a semicolon operator; checks for a DO-initialize operator; if found, purges the index variable from the subscript expression table.
G	Checks for an opening in the subscript expression table.
JAM	Adds a subscript expression to the subscript expression table; tags the variable name with the entry number.
JAY1	Closes up the string using TOT as a counter; adjusts the statement Norm.
JAY2	Clears the subscript expression table; indicates a referenced statement number has been encountered.
MOVE	Moves the statement pointer to the next statement.
OPEN	Opens the statement two words; adds the SGT address; adjust the statement Norm.
OPEN1	Generates the SGT; places it into the Symbol Table; checks for a Symbol Table overlap.
OUT	Transfers to the ROL routine to load the next phase.
PIG	Replaces the erroneous statement by an error message.
PURG	Purges an entry from the subscript expression table.
START	Checks for a Symbol Table overlap; initializes the pointers and switches.
TLR	Clears the entire subscript expression table.
TEST	Checks for END, CONTINUE, DEFINE FILE, CALL LINK or CALL EXIT, and FORMAT statements.
TEST1	Checks for referenced statement numbers; checks for the special statement reference, the generated-label.
TEST2	Checks for no entries in the bound variable table.
TST4	Gets the subscript expression from the string; checks for a duplicate entry in the subscript expression table.
TST5	Tags the variable name with the subscript expression table entry number; checks for a literal subscript.
TST5	Removes symbols from the statement using Y as a counter.
TOR2	Purges common variables from the subscript expression table.

Phase 17: Scan

Chart: DV

- Converts all READ, FIND, WRITE, IF, GO TO, CALL, statement function, and arithmetic statements into modified Polish notation.
- Establishes the order of operational performance.
- Sets up the arguments for subroutine calls to be generated.

The Scan Phase converts all READ, WRITE, GO TO, arithmetic, statement function, CALL, and IF statements to a modified Polish notation. This conversion is accomplished through the use of a forcing table, strings, and an Interpreter.

The forcing table is a table of 2-word entries. The first word contains the left and right forcing values for each operator. The first 8 bits comprise the left forcing value and the last 8 bits, the right forcing value. The second word of the 2-word entry

contains the address of the string to be used by the Interpreter when the corresponding operator is forced.

The string address for each operator is a pointer used by the Interpreter. This pointer designates to the Interpreter a string of operations which must be performed by the Interpreter in order to convert the forced operator and its operands to modified Polish notation.

NOTE: Strings sometimes contain pointers also. These pointers designate substrings, which are detailed extensions of the string and which are used by the Interpreter in the same manner as the strings.

A forcing condition exists if the forcing value of the right operator is equal to or greater than the forcing value of the left operator. This condition results in the left operator being forced; that is, the operation to the left has precedence. Whenever a non-forcing condition exists the operands and operators involved remain in the statement. Operands and operators are removed from the string only when an operator is forced and the Interpreter-generated symbol FAC replaces them.

The Interpreter controls the conversion of the statement to Polish notation. As each operator is forced, the Interpreter, using the string address from the forcing value table, selects the associated string and performs the string operations. These operations result in the outputting of the forced operator and its operands, re-sequenced in the order of operational performace. The forced operator and its operands are outputted into the output buffer by the Interpreter and are replaced in the statement body by the symbol FAC.

The scan begins with the string pointer moving from left to right. When an operator is encountered, the scan looks two words to the left for a second operator. If none is found, the string pointer moves right one word at a time in search of another operator. If an operator is found to the left, the scan converts the left and right operators to their respective forcing values and checks for the forcing condition.

If a forcing condition does not exist the scan again resumes, moving the pointer to the right. If, however a forcing condition does exist, the Interpreter handles the operator and operands involved.

Upon return to the scanning process, the string pointer is positioned to the same operator which caused the previous force, the symbol FAC resides one word to the left in place of the forced operator and its operands, and a new operator resides two words to the left. These operators are then converted and the check is made for the forcing condition.

If, at any time, the symbol FAC is an operand of a forced operator, FAC is replaced by a GT (generated-temporary storage location). The GT is then outputted as the operand in place of FAC. FAC again replaces the forced operator and operands in the statement body. New GTs are created as they are needed in order to maintain FAC in the statement body.

At the completion of the scan process the statement body has been reduced to the symbol FAC; the statement body now consists of less than four words. The output buffer contains the entire statement converted to the modified Polish notation.

If, in looking for a left operator, the scan must bypass the argument list of a call operator, the elements of this argument list are stored temporarily in a special buffer called the push-down list. When the call operator is forced and placed into the output buffer, the push-down list is then emptied into the output buffer in reverse order so that the arguments are restored in their original sequence following the call operator.

When the scan detects that the statement consists of less than four words (the symbol FAC only) the output buffer is placed into the statement string, overlaying the symbol FAC, and the scan moves to the next statement.

The statement terminator serves as an operator which is scanned as any other operator.

See Figure 16 for an example of the scanning process.

Errors Detected. The following error is detected in this phase: 64.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 17.

Routine/ Subroutine	Function
BADST	Replaces the erroneous statement by an error message; closes up the string.

Arithmetic Statement
A = B + C * D - E

Step	Contents of the string	Contents of the Output Buffer	Comments
1	A = B+C*D-E ↑ P		A is not an operator, so the pointer, P, moves to the right. When scanning to the right for an operator, non-operators are simply skipped. When scanning for an operator to the left with which to force, non-operators are added to the push-down list.
2	A = B+C*D-E ↑ P		There is no operator to the left with which a forcing condition test can be made; therefore, the pointer moves to the right until an operator is encountered.
3	A = B+C*D-E; ↑ P		Using the forcing table, the pointer indicates the RV (right forcing value) and P - 2 (two positions to the left) indicates the LV (left forcing value). The RV of + is 0A; the LV of = is 3B. 0A is not equal to or greater than 3B. The left operator is not forced, and the pointer moves to the next operator.
4	A = B+C*D-E; ↑ P		The RV of * is 05; the LV of + is 0A. 05 is not equal to or greater than 0A. The left operator is not forced, and the pointer moves to the next operator.
5	A = B+C*D-E; ↑ P	*CD	The RV of - is 0A; the LV of * is 05. 0A is greater than 05. Hence, the left operator is forced. *CD is placed in the output buffer and the symbol FAC replaces the outputted operator and operands. The pointer is not moved; an attempt is made to force the new left operator.
6	A = B+FAC-E; ↑ P	*CD+B	The RV of - is 0A; the LV of + is 0A. 0A is equal to 0A. Hence, the left operator is forced. +B is added to the output buffer. The symbol FAC still replaces the contents of the output buffer. The pointer is not moved; an attempt is made to force the new left operator.
7	A = FAC-E; ↑ P	*CD+B	The RV of - is 0A; the LV of = is 3B. 0A is not equal to or greater than 3B. The left operator is not forced and the pointer moves to the next operator.
8	A = FAC-E; ↑ P	*CD+B-E	The RV of ; is 3C; the LV of - is 0A. 3C is greater than 0A. Hence, the left operator is forced. -E is added to the output buffer. The symbol FAC still replaces the contents of the output buffer. The pointer is not moved; an attempt is made to force the new left operator.
9	A = FAC; ↑ P	*CD+B-E=A	The RV of ; is 3C; the LV of = is 3B. 3C is greater than 3B. Hence, the left operator is forced. =A is added to the output buffer. The symbol FAC still replaces the contents of the output buffer.
10	FAC;	*CD+B-E=A	The original statement now consists of three words or less. This indicates that the statement has been scanned. The symbol FAC is now replaced by the contents of the cutput buffer.
11	*CD+B-E=A;		The scan is complete.

Figure 16. Scan Example

Routine/ Subroutine	Function
CKNM	Check the statement norm for a count of less than four.
CLOSE	Closes up the statement string.
COMGT	Computes a GT (generated-temporary storage location) in place of the symbol FAC.
CRTNM	Adjusts the statement Norm.
DELET	Deletes the number of symbols specified by the operand.
FORCE	Changes operators to scan-forcing code; checks for a forcing condition.
GOSS	Transfers to the ROL routine to load the next phase.
MOVE	Places the output buffer into the string, deleting the symbol FAC.
MVPT	Moves the pointer by adding the operand to index register one.

Routine/ Subroutine	Function
NAME	Determines if the symbols being pointed at are operators.
NEXTS	Moves to scan the next statement; determines if the last statement scanned was a statement function statement.
OPEN	Determines if the string can be opened; if so, opens the string.
OUTFC	Generates the output buffer under control of a substring; generates GTs (generated-temporary storage locations) as necessary.
OVCHK	Checks for an overlap error in the output area.
PLACE	Stores an operand on the string; if operand is FAC, stores the address of FAC.
PTO	Moves the push-down list to the output buffer until a stop code is detected.
PTPOL	Calculates the statement Norm; determines if the string must be opened or closed.
PUSH	Places the string elements specified by the substring onto the push-down list.
SCANI	Initializes to scan the statement body.
SETUP	Checks for READ, WRITE, IF, GO TO, CALL, statement function, and arithmetic statements; checks for the END statement.
START	Checks for a Symbol Table overlap; initializes the pointers and table areas.
STOPD	Deletes the stop code from the push-down list.
STOP1	Inserts the stop code into the push-down list.
SUBSC	Outputs subscript elements unchanged.

Phase 18: Expander I

Chart: DW

- Replaces READ, FIND, WRITE, GO TO, and RETURN statements with compiler-generated coding.
- Sets up implied DO-loops within READ and WRITE statements.
- Replaces those parts of arithmetic, IF, CALL, and statement function statements that involve subscripting of variables with compiler-generated coding.
- Checks subprograms for a RETURN statement.

Phase 18 replaces READ, WRITE, and FIND statements by a call to the appropriate I/O routine along with the necessary arguments. Generated-labels are added to READ and WRITE statements involving implied DO-loops. These special labels are handled in Phase 22.

Also, statement function, arithmetic, IF, and CALL statements are examined for subscripted variables. Those parts of these statements which involve subscripts are replaced by compiler-generated coding.

In order to produce more efficient coding, a table is created of the SGTs (subscript-generated-temporaries) generated in Phase 16.

By means of the SGT table unnecessary 'LDX L1' and 'LDX I1' instructions are avoided. Also, 'LDX L1' instructions used for literal subscripts are placed immediately before the indexed operations. During the process of macro-expansion, SGT indicator bits are removed from subscripted variables.

In the case of a FUNCTION subprogram, the name of the program is loaded in output format into FNAME in the FORTRAN Communications Area. Also, RETURN statements in FUNCTION subprograms are converted to an accumulator load sequence followed by a 'BSC I', rather than the simple 'BSC I' used for SUBROUTINE subprograms.

Errors Detected. The following error is detected in Phase 18: 69

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 18.

Routine/ Subroutine	Function
D1011	Detects the disk I/O operator; outputs the appropriate operators and arguments.
EXIT	Transfers to the ROL routine to load the next phase.

Routine/ Subroutine	Function
OUTP	Moves a word from the accumulator to the output string; checks for a Symbol Table overlap.
P1061	Generates 'CALL *COMP' when a WRITE statement is detected.
P2011	Produces a 'BSC L' instruction from a simple GO TO operator; if tracing is required, outputs the call to the trace routine.
P2012	Produces a 'BSC I1' instruction from a computed GO TO operator; if tracing is required, outputs the call to the trace routine.
P2031	Generates the return linkage from a subprogram; allows for the arguments to be passed.
P3031	Identifies the list variable type.
P3041	Generates the calls for dimensioned list variables with the associated SGT.
P3043	Generates the calls for non-dimensioned list variables with the associated SGT.
P3051	Generates the call for a dimensioned list variable without the associated SGT.
P4011	Generates DO-initialize code.
P4023	Generates DO-test code.
PASSA	Moves the pointer past the arguments of 'LIBF SUBSC'.
Q1021	Initializes to scan the next statement.
Q1022	Extracts the statement type.
Q1031	Tests for arithmetic, statement function, CALL, IF, GO TO, READ, FIND, WRITE, and RETURN statements.
Q1041	Retains the current statement in the string unaltered.
Q1051	Outputs the name and dummy variables for statement function statements.
Q2011	Identifies the next operator.
Q4011	Scans the 2-word call argument list for subscripted names; generates the instructions for object-time address insertion for subscripted variables.
Q4021	Outputs the 2-word subprogram call with arguments.
Q5011	Computes displacement values for 'STX L1' instructions; inserts those values in the second word of the instruction.
SBSLT	Makes the literal subscript entry into the SGT table.
SBSN	Generates the call for subscript calculation with arguments; makes the SGT table entry.
SCHP2	Generates the 'LDX' instruction, using the SGT table.
START	Moves the statement string next to the Symbol Table; checks for a Symbol Table overlap.
SYMT	Gets the ID word of a Symbol Table entry.
TMTE	Inserts the function name into FNAME in the FORTRAN Communications Area in output code.

Phase 19: Expander II

Chart: DX

- Replaces arithmetic, statement function, CALL, and IF statements not involving subscripted variables by compiler-generated coding.
- Completes the replacement of arithmetic, statement function, CALL, and IF statements that do involve subscripted variables by compiler-generated coding.
- Optimizes IF statement branch instructions.
- Handles mixed-mode arithmetic.

This phase generates the code necessary to replace arithmetic, statement function, CALL, and IF statements. This phase wholly converts statements of these types that include no subscripted variables and merely completes the conversion, which was partially completed in Phase 18, of statements of these types that do include subscripted variables.

Phase 19 generates the code to perform integer, real, and mixed-mode arithmetic. Where possible integer arithmetic is done in-line. The remainder

of the coding consists of calls to System Subroutines followed by argument lists.

As needed, calls to the System Subroutines IFIX and FLOAT are generated. All calls to these subprograms are made 1-word calls. However, calls to exponentiation subroutines generated by Phase 19 are made 2-word calls.

GTs (generated-temporary storage locations) detected in the string by this phase or generated by this phase for storing intermediate results of arithmetic calculations are made to agree in mode with the function of which they are a part.

IF statements are optimized to combine branch instructions when a statement number appears more than once in an IF statement. Also, a branch to an immediately following statement is omitted from an IF statement.

Errors Detected. There are no errors detected in Phase 19.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 19.

Routine/ Subroutine	Function
CAXB	Makes the Symbol Table entry for the subroutine names FAXB and EAXB.
CAXBX	Makes the Symbol Table entry for the subroutine names FAXBX and EAXBX.
D5011	Outputs disk READ/WRITE statements.
FINDN	Locates the next operator not yet processed.
GETGT	Computes the GT Symbol Table address.
GTMOD	Makes the mode of the GT agree with the current mode of modeswitch.
OUTUN	Adds a word to the statement string from the accumulator.
OUTNA	Adds a name to the statement string packed into one word with an operator.
P1021	Initializes to scan the next statement; extracts the statement ID type.
P1031	Checks for a CALL, IF, arithmetic, or statement function statement.
P1041	Moves the statement to the output string unaltered; determines if the last statement was an END statement.
P1051	Outputs the name and dummy variables from the statement function statement.
P2011	Moves the pointer past the processed part of the string entry; retains the processed part in the string unaltered; picks up at the next not processed word.
P2031 thru P2091	Generates the floating point or fixed point arithmetic specifications.
P3011	Checks for an IF operator; generates the tracing calls; checks for a statement number on the next statement; optimizes the 'BSC L' instruction generated to combine conditions.
P4011	Moves a subscript expression and its arguments to the output string.
P5011	Detects a call operator; sets modeswitch; if IFIX or FLOAT call, changes it to a 1-word call.
P5023	Checks for the EXTERNAL specification; generates the proper call; outputs the complete call and arguments.
P6011	Generates the calls or instructions to handle the unary minus.
REVOP	Changes the subtract, divide, and exponentiate operators into reverse operators where needed.
SCKLD	Generates the code for a 'LD' instruction in the designated mode.
MOVST	Moves the string next to the Symbol Table.
STENT	Gets the name of the next new Symbol Table entry.
SYMT	Gets the Symbol Table ID word of the variable name.

Phase 20: Data Allocation

Chart: DY

- Allocates storage for COMMON variables.
- Allocates all storage assignments aligned

according to EQUIVALENCE Statements.

- Assigns all allocations according to the specified precision of the program.

- Prints the allocations of the variables as they are assigned, if requested.

Phase 20 performs the function of allocation of the variables found in the Symbol Table; i.e., these variables are assigned object-time storage addresses. This object-time address replaces the Symbol Table entry of the variable.

The COMMON Area at object time resides in high-addressed storage. Thus, all COMMON variables are assigned absolute addresses within this high-addressed area. The variable area at object time resides in storage just below the object program. All variables not in COMMON are assigned relative addresses within this area.

Phase 20 first allocates COMMON variables found in the Symbol Table. EQUIVALENCE statements that include COMMON variables are then examined and the addresses are aligned during allocation to obtain an equivalence. A check is made to ensure that the equivalence does not extend beyond the limits of the COMMON area.

Variables that appear in EQUIVALENCE statements are allocated next, one combined equivalence nest at a time. A check is made to determine that one variable in a combined nest is defined. The remaining not yet allocated variables in the Symbol Table are finally allocated, real variables first, followed by integer variables.

Undefined variables are not allocated (see Phase 21).

The printing of the Symbol Table variables takes place during Phase 20, if the option has been specified in a control record.

Phase 20 also computes the core requirements for constants after all defined variables have been allocated. The core requirements for variables and for COMMON are then stored in the FORTRAN Communications Area.

Errors Detected. The error detected by Phase 20 is either: 65, 66, or 67.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 20.

Routine/ Subroutine	Function
ARRL	Computes the size of an array.
DEFVT	Determines if the variable is defined; if defined, sets an indicator.
EVENA	Makes the variable area start at an even address if the defined precision is standard.
GETD4	Gets the value of D_4.
INSER	Inserts the "allocated" bit and the "defined" bit into the statement ID word. Moves to the print area the variable allocation, if a listing is required.
PRINT	Performs the printing of the data allocation on the system printer.
NEXT	Bypasses I/O calls and error messages in the string.
P821A	Sets the integer variable format according to the required integer format.
P821C	Checks for EQUIVALENCE statements; detects error messages and I/O calls.
P8221	Checks the Symbol Table entry of a variable for the COMMON indicator.
P8222	Stores the allocation of the encountered COMMON variable as the alignment value of the current nest.
P8253	Checks that exactly one item in the nest was previously allocated.
P8311	Begins processing the allocation of equivalenced items that are not in COMMON.
P8331	Marks the first nest in a combined nest by inserting allocation bits into the Symbol Table entry of the involved variables.
P8411	Begins identifying the remaining nests in a combined nest.
P8412	Checks whether the next equivalence nest contains any connecting variables, thus belonging to the current combined nest.
P8431	Establishes the alignment of the new nest (nest alignment equals the nest alignment of the previously

Routine/ Subroutine	Function
	identified nest, corrected with D_4 in case the connecting variable is dimensioned).
P8441	Inserts the nest alignment value into the new nest to be included in the combined nest.
P8442	Checks that there is only one connecting variable in the new nest.
P8512	Allocates all variables in the previously identified combined nest.
P8523	Locates the next equivalence nest within a combined nest.
P8524	Determines if the next equivalence nest is included in the current combined nest.
P8531	Modifies the size of the variable area by the size of the combined nest just allocated.
P8621	Allocates COMMON variables, making the names right-justified.
P8641	Allocates real variables.
P8651	Allocates integer variables.
P8711	Inserts the work area size and the constants area size into the FORTRAN Communications Area.
TCOMB	Tests for the combined nest limits.
VARFO	Computes the variable format.

Phase 21: Compilation Errors

Chart: DZ

- Lists any and all errors that were detected during the compilation process.
- Rearranges the statement string, if there were no errors detected.

The Compilation Error phase initially scans the statement string, deleting EQUIVALENCE statements that do not have an error indicator and replacing EQUIVALENCE statements that have an error indicator by error messages.

The Symbol Table is then scanned twice. The first scan detects unreferenced numbered statements and lists them on the system printer. The second scan detects undefined variables and lists these also on the system printer. If an undefined variable is detected, an indicator bit (bit 14) is set in the ERROR word in the FORTRAN Communications Area. The presence of the bit causes the suppression of the final output.

The statement string is scanned for error messages. Two counters are maintained during this scan. The first (STLAB) contains the statement number of the last numbered statement encountered. The second (STCNT) contains a count of the statements encountered since the last numbered statement. Compiler-generated statements and statement numbers are disregarded in these counts.

When an error message is detected, these counters are inserted into the error message along with the error number for printing. In addition, bit 14 of the ERROR word in the FORTRAN Communications Area is set in order to inhibit the final output.

After the printing of the error messages, bit 15 of the ERROR word is tested. If this bit is on, an overlap error has occurred during the compilation. Phase 21 then prints the messages "Program length exceeds capacity" and immediately terminates. If the overlap error indicator (bit 15) is not on, bit 14 of the ERROR word is tested. If this bit is on, the message "Output has been suppressed" is printed and the phase is terminated.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 21.

Routine/ Subroutine	Function
A1011	Searches the Symbol Table for unreferenced numbered statements.
A1021	Searches the Symbol Table for undefined variables.
CONV	Converts the characters into the output code of the system printer.
CON3D	Converts the statement number counter and error type value from binary mode to decimal, both 3 digits.
ERRSW	Sets the ERROR switch in the FORTRAN Communications Area to inhibit the final output.
E1021	Searches for the Error statements

Routine/ Subroutine	Function
	on the statement string.
MOVS1	Initializes to move the statement; checks for an EQUIVALENCE statement; checks for error marks; deletes EQUIVALENCE statements with error marks.
MOVS2	Inserts the Error message for EQUIVALENCE statements with error marks.
MOVS3	Checks for the END statement; if not the END statement, retains the statement in the string.
MOVS5	Stores the end-of-string address.
PRINT	Performs the printing on the system printer.
R1011	Rearranges the statement string, placing the DEFINE FILE, statement function, and FORMAT statements at the low-address end of the string.
START	Initializes the phase.

Phase 22: Statement Allocation

Chart: EA

- Assigns the relative addresses to statement functions and numbered statements; inserts the allocations into the string.
- Creates the subroutine initialization call, if required.
- Calculates the core requirements of the program; stores the result in the FORTRAN Communications Area.
- Generates the statement function return code.

Phase 22 allocates relative addresses to all numbered statements and statement functions. The allocation is placed into the statement string entry, following the statement number or function label.

From the Location Counter at the end of allocation, a calculation of the program's storage requirements is made and stored in the SOFNS word in the FORTRAN Communications Area.

If the program being compiled is a subprogram, this phase also creates the subroutine initialization call, 'CALL SUBIN', along with its psuedo-arguments, which directs the insertion of arguments at object time. In the case of a FUNCTION subprogram, Phase 22 also generates the accumulator load and branch instructions which comprise the return linkage to the mainline program.

Errors Detected. There are no errors detected by this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 22.

Routine/ Subroutine	Function
ENT	Checks for a Symbol Table overlap; moves the string adjacent to the Symbol Table.
GETST	Gets the ID word of a Symbol Table entry.
LDR	Moves a DEFINE FILE statement to the output string, not counting its locations.
MOVST	Moves a string item from the input string to the output string.
M1011	Adds the size of the work area to the size of the constants area; equates the Location Counter to this value.
M1021	Extracts statement Norm and ID type; checks for FORMAT and I/O call types.
M1031	Moves FORMAT and I/O call statements to the output string unaltered; allocates the statements; eliminates the labels of these statements from the string.
M1051	Allocates a statement function statement.
M1071	Allocates a statement other than a FORMAT or statement function statement.
M1081	Checks for dummy variables in the Symbol Table; outputs them behind the 'CALL SUBIN'.
NAMT	Eliminates duplicate names from the same argument list.
OUTP	Moves a word from the accumulator

Routine/ Subroutine	Function
	to the output string; checks for a Symbol Table overlap.
SUBPR	Checks for a subroutine name in the call arguments; if required, adjusts argument list to allow for the subroutine name.
S2011	Initializes statement function statements, initializes subprograms, or allocates statements and moves the string, depending on the value of TRACKSWITCH.
S2165	Checks for 'CALL SUBIN'; if found, moves pointer past argument list; counts the locations.
S2172	Moves 'CALL SUBSC' and its SGT and D_4 arguments to the output string.
S2182	Moves the subscripted variable and the D_1, D_2, and D_3 arguments to the output string.
S2201	Generates a 'BSC L' instruction from a generated label operator.
S3013	Generates 'CALL SUBPR' with name if the name following a call operator is a dummy variable or external name.
S3015	Generates a 'BSI L' instruction if the name following a call operator is both a dummy variable and external name.

Phase 23: List Statement Allocations

Chart: EB

- Inserts the statement allocations into the Symbol Table.
- Lists the statement allocations on the system printer.

This phase scans the statement string for statement function statements and numbered statements. The allocation found in each of these statements is entered in the Symbol Table. The allocation and the label are then deleted from the string entry. The remainder of the numbered statement or statement function statement is moved to the output string unaltered.

If specified in a control record, the statement allocation and statement number or statement function name are listed on the system printer.

Errors Detected: There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 23.

Routine/ Subroutine	Function
ALOC	Gets the Symbol Table entry; checks for a list request; bypasses generated labels.
ENT	Initializes input and output string pointers; checks for an overlap error.
EXIT	Transfers to the ROL routine to read in the next phase.
LIST	Converts the statement label and allocation address to output code.
MOVE	Moves the statement to the output string unaltered.
M1000	Checks for a numbered statement or statement function.
PRINT	Performs the printing on the system printer.

Phase 24: List Symbols

Chart: EC

- Lists the Features Supported by the program as indicated in the FORTRAN Communications Area.
- Lists the System Subroutines used by the program, if requested.
- Lists the subprogram names found in the Symbol Table, if requested.
- Makes additions to the coding generated by Phases 18 and 19, Expander I and II.

Using the indicators in the CCWD word in the FORTRAN Communications Area, Phase 24 re-creates the control records which were recognized by the compiler in Phase 1. These control records,

excepting the *IOCS, are listed on the system printer under the title 'Features Supported'.

According to the indicators in the CCWD word, Phase 24 also alters, for purposes of printing, the names of subprograms in the compiler-generated calls to reflect Extended Precision, if specified. The actual compiler-generated coding is not altered until Phase 27.

If requested, a list is made of all the subprogram names that appear in the Symbol Table.

Phase 24 also scans the statement string, bypassing all single word statements and tagging the names in the System Subroutine table which are called by the program. The Subroutine table is then scanned and, if requested, the tagged System Subroutine names are listed.

As a supplement to the Expander phases, 18 and 19, Phase 24 adds indexing to System Library Subroutines and to 2-word instructions which reference dimensioned variables.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 24.

Routine/ Subroutine	Function
CHNAM	Changes the names of called subprograms to reflect Standard or Extended Precision specifications.
CNVAD	Converts the address in the accumulator into printer code.
CONV	Converts characters to printer code.
ENT	Initializes the system printer; checks for a Symbol Table overlap.
EXIT	Transfers to the ROL routine to load the next phase.
GETST	Gets the ID word of a Symbol Table entry.
K1031	Prints 'IOCS', if applicable.
K1021	Prints 'Arithmetic Trace' if applicable.
K1035	Prints 'One word Integers' if applicable.
K1041	Prints 'Save Loader' if applicable.
K1051	Prints 'Extended Precision' if applicable.
L1035	Prints 'Called Subroutines'.
L1041	Initializes for the scan of the Symbol Table; checks for a constant; compares subprogram names to the compiled program name; compares statement function names to the compiled program name; gets the compiled program name from the Symbol Table.
L2021	Checks for the END statement; checks for one word statements and FORMAT statements.
L2031	Adds indexing to packed instructions if the name refers to a dimensioned variable.
L2033	Checks for the special 'BSC L' instruction; adds an extra word to it.
L2041	Checks for a Symbol Table list request.
L3011	Checks for a 2-word System Subroutine or subprogram call; adjusts call codes involving dimensioned variables.
L3013	Checks for the special argument list on subroutine calls; moves the pointer past the argument list.
L3051	Checks for a call to a statement function; changes these calls to 'BSI L' instructions.
MNAME	Converts the subprogram name to printer code; moves it to the print area.
PRINT	Performs the printing on the system printer.
START	Initializes the transfer vector; prints 'Features Supported', if applicable; prints 'Transfer Trace' if applicable.

Phase 25: List Constants

Chart: ED

- Lists the Core Requirements.
- Lists the constants and their addresses if requested.

Under the heading 'Core Requirements' Phase 25 prints the amounts of storage used by the program, COMMON area, and variables. The program name is printed from the FNAME word in the FORTRAN Communications Area.

If a list request is specified in the CCWD word, the real and integer constants are converted to output code and listed with their relative addresses according to the specified precision. Real constants are listed first, followed by integer constants.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 25.

Routine/ Subroutine	Function
CONVI (TOPAU)	Converts an integer constant to the printer code; moves the constant to the print area.
COREQ	Prints the core requirements of the program.
ENT	Initializes the phase; checks for a Symbol Table overlap.
EXIT	Transfers to the ROL routine to load the next phase.
L4011	Prints 'Core Requirements'; if requested, sets up to list the constants; checks for Extended Precision; makes modifications in the constant formats if Extended Precision.
MULT1	Converts the binary constant to decimal.
PRINT	Performs the printing on the system printer.

Phase 26: Output I

Chart: EE

- Builds the program header and data header records.
- Places these records onto the disk in Working Storage.
- Places real and integer constants into the Working Storage in absolute mode.

Phase 26 initially builds the program header and data header records. These records and the buffer communications area carry pertinent information about the compiled program to Phase 27. The program header and data header records are placed in Working Storage.

The statement string is searched for DEFINE FILE statements. These statements are analyzed and then placed into Working Storage. The file specifications are outputted in absolute mode, except for the associated variable which is in relocatable mode (the only relocatable constant).

The Symbol Table is scanned twice. The first scan extracts real constants, computes their allocations, and inserts the allocations into the Symbol Table. The second scan performs the same operations for integer constants. All constants are placed into Working Storage in absolute mode.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 26.

Routine/ Subroutine	Function
EXIT	Transfers to the ROL routine to load the next phase.
Q1014	Checks for more than one DEFINE FILE statement.
GETST	Gets an address or name at object time from the Symbol Table.
OUABS	Outputs an address or constant in absolute mode.
OUREL	Outputs an address or constant in relocatable mode.
Q1011	Checks for a mainline program or a subprogram; sets additional information into the output area; outputs a DEFINE FILE statement.
Q1022	Outputs real and integer constants in the absolute mode.
Q1033	Inserts a constant allocation into the Symbol Table.
START	Sets disk I/O indicators into the output area; stores the FNAME

Routine/ Subroutine	Function
	word in the output area; checks for the first DEFINE FILE statement.
WRITE	Writes the output buffer onto the disk in Working Storage.

Phase 27: Output II

Chart: EF

- Converts the compiled statement string to output code.
- Places the object program into Working Storage.

According to the indicators in the CCWD word in the FORTRAN Communications Area, Phase 27 alters the subroutine names referenced by the compiled program, to reflect, if necessary, the Extended Precision as specified by the user. Phase 24 made the same conversion for listing purposes; this phase makes the conversion during the generation of the object program.

Phase 27 then converts the statement string into object code. The object code is placed onto the disk in Working Storage.

At the completion of the output, the termination routine (OUTER) inserts the necessary data into the FORTRAN Communications Area so that the Recovery Phase can complete the compilation.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 27.

Routine/ Subroutine	Function
CHNAM	Changes the names found in the Subroutine table.
GETST	Gets an object time address or name from the Symbol Table.
INIT	Initializes to scan the next statement.
OUABS	Outputs an address or constant in absolute mode.
OUREL	Outputs an address or constant in relocatable mode.
OUTER	Sets the final indicators; transfers to the ROL routine to load the last phase.
Q2012	Checks for FORMAT, DEFINE FILE, CALL LINK, and CALL EXIT statements.
Q2211	Outputs 'BSI L' with name for a CALL LINK or CALL EXIT statement.
Q4031	Outputs an absolute value following an 'LDX L1' instruction.
Q4041	Outputs an entry address (an object time linkword) following a 'BSC I' instruction.
Q7011	Outputs a one-word call for System Subroutine calls.
Q7021	Converts one word to object time instruction; outputs in absolute mode.
Q8011	Outputs the arguments for 'Call SUBSC'.
Q8051	Outputs arguments of calls FIOAX and FIOIX.
START	Checks for Extended Precision; alters Subroutine names, if necessary, to reflect precision.
T7005	Outputs a two word call.
T8013	Outputs addresses in relocatable mode; if variable is in COMMON, outputs in absolute mode.
TOBUF	Moves a word to the output buffer.
WRITE	Writes the output buffer onto the disk in Working Storage.

Phase 28: Recovery

Chart: EG

- Restores the Skeleton Supervisor from the CIB.
- Sets up the switches and parameters needed by the Skeleton Supervisor to assume control.

This phase handles the return to the Skeleton Supervisor following completion or termination of the compilation.

An entry at ENT1 results from the detection of a Monitor control record during the Input Phase.

An entry at ENT2 results from an excessive number of disk errors occurring during compilation or from a Working Storage overflow.

An entry at ENT3 results from a direct Monitor call, interrupting the compilation, manually executed by the machine operator.

An entry at ENT4 results from a normal end of compilation condition, with or without errors.

In each case the Recovery Phase sets indicators in the FORTRAN Communications Area in order to inform the Skeleton Supervisor as to the results of the compilation.

Disk errors, compilation errors, the exceeding of Working Storage, and the trapping of a Monitor control record all cause the compilation output to be suppressed, the non-XEQ switch to be set, and the non-DUP switch to be set.

If the compilation is successful, the program length, the number of disk blocks used to store the program, and the XEQ address are all transmitted to the Skeleton Supervisor.

Errors Detected. There are no errors detected in this phase.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in Phase 28.

Routine/ Subroutine	Function
ENT1	Disables the end of compilation message; inhibits program execution.
ENT2	Prints the message 'Over 50 disk errors at sector xxxx'; inhibits program execution.
ENT3	Inhibits program execution; prints the message 'MON Call'. Checks for compilation errors; if none, sets up for program execution.
GMON	Restores the disk interrupt word; prints the message 'End of Compilation'.
RCMON	Recalls the Skeleton Supervisor from the Core Image Buffer on the disk.
RTN	Transfers control to the Skeleton Supervisor.

Phase: Dump

Chart: EH

- Dumps the FORTRAN Communications Area, statement string, and Symbol Table upon request.

The ROL routine, prior to loading the next phase, calls the DUMP routine. This routine tests the status of the Console Entry Switches. If the number entered in the switches equals the number of the phase just completed, the DUMP routine initializes to load the next phase but calls instead the Dump Phase.

The Dump Phase prints the contents of the statement string, the FORTRAN Communications Area, and Symbol Table onto the 1132 Printer. At the completion of the dump, the Dump Phase returns to the ROL routine which loads the next compiler phase.

Special User Information. The Dump Phase is included in the Monitor System FORTRAN Compiler. However, since inter-phase dumps are not normally required, the call to the Dump Phase is bypassed.

In order to implement the Dump Phase, the user must make the following modification to the ROL routine in Phase 1:

the two instructions

```
GODMP   STX 1 ROL3
        STX 2 CNT
```

must be deleted and the following instruction inserted:

```
GODMP BSC L DUMP
```

This change will enable the inter-phase dump feature.

Errors Detected. If the dump address-parameter is a negative quantity, the Dump Phase prints the message

ERROR IN ADDRESS

and terminates the dump.

Routine Summary. The following descriptions summarize the purpose or function of the major routines and subroutines contained in the Dump Phase.

Routine/ Subroutine	Function
DUMP	Checks for a valid address-parameter; if valid dumps the contents of the FORTRAN Communications Area, statement string, and Symbol Table as per the parameters given by XQX1.
HEX	Changes storage addresses to hexadecimal print code.
LOADR	Halts; transfers to the ROL routine when START is pressed.
PRINT	Performs the printing of the dump according to the formats given by DUMP.
XQX1	Sets up the print counters and indicators; computes the address-parameters for the statement string and Symbol Table.

This section contains a discussion of a representative cross-section of the I/O subroutines. For details regarding any of the subroutines contained in the Subroutine Library, see the publication IBM 1130 Subroutine Library (Form C26-5929).

CARD SUBROUTINE (CARD1)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered via an LIBF statement, the contents of XR1, XR2, the Accumulator, and the status indicators are saved. The control parameter is then checked to determine the type of function requested.

If a Test function is specified, the routine Busy indicator is checked and the saved registers are restored. If the routine is busy, the subroutine exits to LIBF+2; if it is not busy, to LIBF+3.

An error exit is made to location 41 if the card read punch is not ready (and not busy) or if a parameter or the word count is in error. If the card read punch is not ready, the error indicator is on, and the requested function is Read or Feed, a skip indicator is set before the exit to location 41 is executed. If the card read punch is not ready and busy, the subroutine loops until the 1442 is ready.

If a Read function (GET) is specified, a read IOCC is built for use by the column interrupt routine. Restart data (word count and I/O address) is saved and each word of the input area is initialized to one (/0001) for use in determining if a column has been read. If the last card indicator is on, the card is ejected (but causes no interrupt) and an error exit to location 41 for "not ready" is made. The ISS counter is incremented by one, and the routine Busy indicator is set. If the skip indicator is set, indicating a previous feed check error, a feed operation is initiated to pass the first card through the read station. If the skip indicator is off, a read operation is initiated. The saved registers are then restored and the routine exits to LIBF+3.

If a Punch function (PUT) is specified, a punch IOCC is built for use with the column interrupt routine and the restart data is saved. The ISS counter is incremented by one, the routine Busy indicator is set, and a punch operation is initiated. The saved registers are then restored and the routine exits to LIBF+3.

If a Feed function (FEED) is specified, the last card indicator is checked. If on, the card is ejected (but causes no interrupt) and an error exit is made to location 41 for a "not ready" condition. Otherwise, the ISS counter is incremented by one, and the routine Busy indicator is set, and a card feed cycle is initiated. The saved registers are then restored and the routine exits to LIBF+2.

If a Stacker Select function (STK) is specified, a select stacker is executed. The saved registers are then restored and the routine exits to LIBF+2.

Column Interrupt Processing

- Determines if last column has been read or punched
- Initiates next read or punch operation

Operation: The word count is decremented by one. If it is not zero, the interrupt indicator is reset, the I/O area address is incremented by one, and another read or punch operation is initiated. The routine then exits to the Interrupt Level Subroutine (ILS00).

If the word count is zero, it is an indication that the last column has been read or is to be punched. If a read operation is being performed, the interrupt indicator is reset, the word count is set to +1 so that when it is decremented it will be zero, and the routine exits to ILS00. If a punch operation is in progress, a last word indicator bit is added to the last character (stored in CHAR) and the punch command is executed.

Operation Complete Interrupt

- Check for errors and last card condition
- Decrement ISS counter and clear busy indicator

Operation: A check is performed for an error, last card, or skip condition. If none is indicated, the

ISS counter is decremented by one, the routine Busy indicator is reset, and an exit to ILS04 occurs.

Last Card: If the last card indicator is on, an exit to the user's Error subroutine takes place. The Accumulator contains the device number and the last card indicator. Upon return from the user's subroutine, the ISS counter is decremented by one, the routine Busy indicator is reset, and an exit to ILS04 occurs. (If the function is Punch, the last card is ejected prior to decrementing the ISS counter.)

Errors: If an error is detected, an exit to the user's Error subroutine takes place. The Accumulator contains the error type code. Upon return from the user's subroutine:

1. If termination is requested (Accumulator = 0), the ISS counter is decremented by one and the routine Busy indicator is reset.
2. If termination is not requested, a loop occurs until the device is made ready. (The operator must clear the reader and then place the nonprocessed cards - or blank cards if punching - back in the reader.) When Reader START is pressed, it is necessary to determine if a feed check occurred, preventing initiation of the operation by the 1442. If the function is feed and the error was not a feed check at the read station, or if the function is read and the first column was not read in, the operation was not initiated. It is necessary to skip the first card replaced in the hopper by the operator and to reinitiate the requested operation on the second card. Therefore the skip indicator is set and a card feed initiated. Otherwise the requested operation is restarted, using previously saved restart data, and an exit to ILS04 occurs without modifying the ISS counter or Busy indicator.

Skip: If the skip indicator is set, the I/O operation causing the operation complete interrupt was a card feed used as part of the error recovery procedure. Therefore the skip indicator is cleared, the operation is restarted, and an exit to ILS04 occurs without modifying the ISS counter or Busy indicator.

KEYBOARD, CONSOLE PRINTER OR OPERATOR REQUEST SUBROUTINE (TYPE0)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered from a mainline program via an LIBF statement, the contents of XR1, XR2, the Accumulator and Extension, and the status indicators are saved.

If a Test function is specified, the routine busy indicator is checked and the saved registers are restored. If the routine is busy, the subroutine exits to LIBF+2; if not busy, to LIBF+3.

If a Read function is specified, the address of the first data word in the I/O area is set into the first word of the Read IOCC. This address is saved at RSTRT+1 for restart. The word count is stored at COUNT and is saved at RSTRT+2 for restart. The addresses for HOLL and PRTY tables are set up for the conversion of the input data. The routine busy indicator is set and the ISS counter is incremented by 1. The Keyboard is then released via an XIO command and the routine exits to LIBF+3.

If a Write function is specified, the word count is doubled because the characters are stored in the I/O area two characters per word. The word count is stored in COUNT. The routine busy indicator is set and the ISS counter is incremented by 1. The word to be printed is stored in a temporary area (TEMP1) and the first character is printed. The routine exits to LIBF+3.

Errors: An exit to location 41 occurs if the device or function is not valid, if the character count is zero or minus, or if the requested device (Keyboard and/or Console Printer) is not ready.

Interrupt Processing

- Determines the cause of the interrupt
- Reads and/or Prints
- Indicates routine and device not busy when the operation is completed

General Interrupt Processing: This routine is entered when an interrupt occurs. It determines the cause of the interrupt and executes a branch to the routine servicing the type of request.

If all characters have been read and/or printed, (character count is 0), the routine busy indicator is reset and the ISS counter is decremented by 1.

If the interrupt is caused by an operator request, the subroutine exits to the address in location 44. (This is the address of a routine that processes operator requests. The user's operator request routine must return to the ISS routine.)

Print Function Interrupt Process: This routine prints data from the I/O area. Data in the I/O area is in

Console Printer code, two characters per word. A switch (RIGHT) is set to zero if the right character (bits 8-15) is to be printed, and to one if the left character (bits 0-7) is to be printed. The address of the word to be printed is incremented by one prior to processing the left character. The character count is decremented by one for each character printed. If the last character has been printed (character count becomes 0), the routine busy indicator is reset, and the ISS counter is decremented by one. Otherwise a character print is initiated. In either case, the routine checks for an operator request before exiting to the ILS.

Keyboard Interrupt Processing: This routine processes input characters from the keyboard.

Graphic Characters: Graphic characters are converted to console printer code and stored in a temporary buffer. If more characters are to be read, the keyboard is made ready. In any case, a character print is initiated, after looping if necessary, until the console printer is ready. The routine checks for an operator request before exiting to ILS04.

End of Message Character: The end of message character is converted to an IBM Card Code NL and stored in the I/O area following the last non-control character read. The routine busy indicator is reset, the ISS counter is decremented by one, and the routine checks for an operator request before exiting to ILS04.

Re-entry Character: When a re-entry character is read, the original address of the I/O area and the word count are restored, a re-entry indicator is set, and a print is initiated to print one slash (/). Processing interrupts while the re-entry indicator is set has the following results: one more slash is printed, the carrier is restored, the keyboard is released, and the re-entry indicator is cleared. After each interrupt, the routine checks for an operator request before exiting to ILS04. The message in the I/O area is not cleared. The new message overlays the previous message, character by character.

Backspace: When a backspace character is read, the address of the I/O area is decremented by one and the word count is incremented by one. The backspace indicator is set, and a backspace initiated. Processing interrupts while the backspace indicator is set has the following results: a slash is printed (over the last graphic character), the backspace indicator is cleared, and the keyboard is released. After each interrupt, the routine checks for an operator request before exiting to ILS04. The last character in the I/O area is not cleared. The next graphic character entered will overlay it.

CONSOLE PRINTER OR OPERATOR REQUEST SUBROUTINE (WRTY0)

- Call processing similar to TYPE0 except that the keyboard cannot be read
- Keyboard interrupts are ignored except for the interrupt request

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered via an LIBF statement, the contents of XR1, XR2, the Accumulator and Extension, and the status indicators are saved.

If a Test function is specified, the routine busy indicator is checked and the saved registers are restored. If the routine is busy, the subroutine exits to LIBF+2, if not busy, to LIBF+3.

If a Write function is specified, the word count is doubled because the words are stored in the I/O area, two characters per word. The word count is stored in COUNT. The address of the first data word is stored in IOAR and the character indicator (RIGHT) is set to indicate that the right character is to be processed next. The character to be printed is put into the temporary buffer (TEMP1). The routine busy indicator is set, the ISS counter is incremented by 1, and a print operation is initiated. The routine exits to LIBF+3.

Errors: An exit to location 41 occurs if the device ID or function, is not valid, or if the character count is zero or minus, or if the Console Printer is not ready.

Interrupt Processing

The interrupt is checked to determine whether it is a print or operator request:

1. A print request causes the character count to be decremented by one and the RIGHT character indicator to be checked. The setting of the RIGHT character indicator determines which

half of the data word is to be processed (words in the I/O area are packed 2 characters per word).
When RIGHT = 0: process the rightmost character (bits 8-15) and set RIGHT to one.
When RIGHT = 1: process the leftmost character (bits 0-7), increment the IOAR address by one, and set the RIGHT indicator to zero.
When the character count goes to zero, the ISS counter is decremented by one. Routine busy is already cleared because the character count was used as the busy indicator. If the character count is not zero, a print operation is initiated. In either case, the routine checks for an operator request before exiting to the ILS.

2. An operator request causes the subroutine to exit to the address stored in location 44. (This is the address of the routine that processes operator requests. The user's operator request routine must return to the ISS routine.) When control is returned, the routine exits to ILS04.

PAPER TAPE SUBROUTINE (PAPT1)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered from a mainline program via an LIBF statement, the contents of XR1, the Accumulator and Extension, and the status indicators are saved. XR1 is set to point to the calling sequence control parameter.

If a Test function is specified, the routine Busy indicator is checked and the saved registers are restored. If the routine is busy, the subroutine exits to LIBF+2, if it is not busy, to LIBF+3.

If a Read or Write (punch) function is specified, the IOCC is built and the CHECK indicator is set if the control parameter specifies that a check of DEL or NL characters is required. The routine Busy indicator is then set and the ISS counter is incremented by one. The Read or Write function is initiated and the routine exits to LIBF+4.

Errors: An exit to location 41 occurs if the device, function, or check digits are not valid, if the character count is zero or minus, or if the requested device (reader or punch) is not ready.

Interrupt Processing (Read)

As each character is read, the character counter, CHAR, is incremented by one. The first character is placed in bit positions 0-7 of the specified word in the I/O area, and the second character is placed in bit positions 8-15 of the same word. Each time CHAR is even, the word count is decremented by one and the I/O area address is incremented by one. When the word count goes to zero, the routine busy indicator is reset, the ISS counter is decremented by one, and the Read function is terminated.

If a check function has been specified, each character read is checked to see if it is a DEL or NL character. When the DEL character is encountered it is not placed in the I/O area and reading continues. When the NL character is encountered, the operation is terminated.

Errors: A read error or a reader not ready condition after the operation is initiated causes an exit to the user's error routine. Read errors are checked as each character is read prior to processing the information.

Interrupt Processing (Punch)

The punching of the first character is controlled by the call processing routine. The interrupt processing routine controls the punching of all remaining characters. The second character of a record comes from bit positions 8-15 of the first word. Whenever a complete word has been punched, the word count is decremented by one and the I/O area address is incremented by one. When the word count goes to zero or when an NL character is encountered, if check is specified, the routine busy indicator is reset, the ISS counter is decremented by one, and the Punch function is terminated.

Errors: A punch not ready condition after the operation is initiated causes an exit to the user's error routine.

PAPER TAPE SUBROUTINE (PAPTN)

- Call processing: similar to PAPT1 except reader and punch can operate simultaneously.
- Interrupt processing routines similar to PAPT1.

Call Processing Routine

- Determines type of device and function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered from a mainline program via an LIBF statement, the contents of XR1, XR2, the Accumulator and Extension, and the status indicators are saved. XR1 is set to point to the calling sequence control parameter and XR2 is set to point to the constants table for the device called (reader or punch).

If a Test function is specified, the routine device busy indicator is checked. If the requested routine is busy, the routine exits to LIBF+2, if it is not busy, to LIBF+3.

For Read and Write functions, separate constant tables are built - one for the reader and one for the punch. Each table contains a device Busy indicator, Check indicator, word count, character count, I/O area address, and error subroutine address.

All other operations are the same as the PAPT1 operation.

Interrupt Processing Routine

Same as PAPT1 operation, except after processing any read interrupt, the subroutine checks and processes any punch interrupt before returning control to ILS04.

PLOT SUBROUTINE (PLOT1)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered via an LIBF statement, the contents of XR1, the Accumulator and Extension, and the status indicators are saved. XR1 is set to point to the calling sequence control parameter.

If a Test function is specified, the routine busy indicator is checked and the saved registers are restored. The subroutine exits to LIBF+2 if the Busy indicator is set, and to LIBF+3 if it is not set.

If the Write function is specified and the Busy indicator is set, the subroutine loops until a not busy condition is indicated. Then, indicators and constants are initialized, and the ISS counter is incremented by one. The Write function to perform the operation indicated by the first hexadecimal digit in the control area character is now executed, the saved registers are restored, and control is returned to the calling program.

Buffers and Indicators

BUF	-	Contains the data word (or partial word) being processed.
BUSY	-	Set to indicate the routine is busy.
DEVIC	-	Contains the number of the plotter being used (zero).
DIGIT	-	Contains the count of the number of characters remaining in the data word being processed.
DUPCT	-	Duplication counter. Contains the count of the number of times a plotter action is to be repeated.
ERR+1	-	Set to the address of the user's error routine.
FIRST	-	Set to indicate that the PLOT subroutine was entered by an LIBF call rather than by an interrupt.
IOAR	-	I/O area address. Initially contains the address of the first data word in the I/O area.
WDCNT	-	Word Count. Contains the count of the number of words in the I/O area.
WORK	-	Contains the hexadecimal character being processed.

Interrupt Processing

- Check for parity error
- Execute next plotter function

Operation: Upon entering the routine from an interrupt, the duplication counter (DUPCT) is checked to determine if the previous plotter action is to be duplicated. This check is accomplished by decrementing DUPCT by one. If it does not change sign or go to zero, the previous operation is repeated and the routine exits to ILS03 routine.

If DUPCT changes signs or goes to zero, it is set to zero and the next plot character is obtained by the GET routine.

GET Routine: DIGIT contains the count of the number of plot characters remaining in the data word being processed. It is set to four each time a new word is entered into BUF. As each hexadecimal

digit is moved to WORK, DIGIT is decremented by one. When DIGIT goes to zero, WDCNT and IOAR are decremented and a new word is placed into BUF for processing.

When WDCNT goes to zero, the ISS counter is decremented by one, the Busy indicator is reset, and an exit to ILS03 occurs.

THE IBM 1132 PRINTER SUBROUTINE (PRNT1)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered via an LIBF statement, the contents of XR1, XR2, the Accumulator and Extension, and the status indicators are saved. The control parameter is then checked to determine the type of function requested.

If a Test function is specified (control parameter zero, i.e., the first hexadecimal digit), the routine Busy indicator is checked and the saved registers are restored. If the device is busy, the subroutine exits to LIBF+2, if not busy, to LIBF+3.

If the first hexadecimal digit of the control parameter is not zero (non-Test function), hexadecimal digits 2, 3, and 4 (bits 4-15) are saved in the Extension and the hexadecimal digit for control (bits 0-3) is checked to determine if the requested operation is Carriage Control (3), Print Alphameric (2), or Print Numeric (4).

If a Carriage Control operation is specified, hexadecimal digits 2 and 3 of the control parameter are used to determine the carriage action. Digit 2 controls immediate actions and digit 3 controls actions after print. The after print switch (AFTIN) is set to the value of hexadecimal digit 2. If its value is zero, it is considered set "on" and the action of the carriage will be determined by the value of hexadecimal digit 3 and will be executed after printing.

The value of hexadecimal digit 2 or 3 is further used to determine whether the carriage should space or skip. If the value is greater than 12, a space is indicated, if one through six, nine, or twelve, a skip. If the operation is a space, the contents of hexadecimal digit 2 or 3 is placed directly in the space or skip counter (SPSK). If the operation is a skip, the contents of hexadecimal digit 2 or 3 is converted, a negative sign is added, and the resultant hexadecimal number is placed is SPSK. The converted number identifies the channel to which the skip will occur. (See chart below.)

If the carriage action is to be executed immediately, (AFTIN not zero) a carriage space or skip is initiated, the ISS counter is incremented by one, the saved registers are restored, and the subroutine exits to the calling program.

If the carriage action is to be taken after printing, (AFTIN zero) the saved registers are restored and the subroutine exits to the calling program.

Value of hexadecimal digits 2 (immediate) and 3 (after print)	Hex value of SPSK	Carriage Action
1	8008	Skip to channel 1
2	8007	Skip to channel 2
3	8006	Skip to channel 3
4	8005	Skip to channel 4
5	8004	Skip to channel 5
6	8003	Skip to channel 6
9	8002	Skip to channel 9
C	8001	Skip to channel 12
D	0001	Space 1
E	0002	Space 2
F	0003	Space 3

If a Print function is specified, the numeric indicator (NUM) will be positive if the output is print numeric, negative for alphameric. Counters will be set for print cycles, words 32-39 will be cleared, bit 15 of word 39 will be set on, and the ISS counter will be incremented by 1. A start print operation (alphameric or numeric) will be initiated and the routine will exit to the calling program.

Errors: An exit to location 41 occurs if any of the following conditions are sensed:

1. An end of forms (or not ready) condition exists
2. An illegal function is being attempted
3. The word count is negative, zero, or over sixty

Interrupt Processing

- Resets the interrupt.
- Tests DSW to determine the function.
- Executes the function.

Operation: A sense and reset command is given to obtain and store the DSW and to reset the interrupt. The stored DSW is then checked to determine if the interrupt is a skip or space response:

If a skip - Compare the channel indicator in the stored DSW with the channel indicator in the SPSK. If they are equal, execute a carriage stop command, decrement the ISS counter by 1, and continue the interrupt processing routine. If they are not equal, the carriage is not at the requested channel and no carriage stop command will be executed.

If a space - If the DSW indicates a space response OR the DSW into PASS. (PASS contains a bit for each channel passed while spacing. It will be checked later to determine if the carriage is at channel 9 or 12.) Decrement the space counter (SPSK) by 1. If the counter goes to zero, decrement the ISS counter by 1 and exit to ILS01. If the counter does not go to zero, execute a space command and continue the interrupt processing. If the DSW does not indicate a space response, check for a character emitter interrupt.

If a Print operation is in progress, the DSW is checked to determine if the print scan has been completed. If it is not completed the routine will exit to ILS01. The following counters are checked during a print operation:

1. Check CTR46. During processing of each character emitter interrupt, CTR46 is tested. If zero, the stored DSW is checked to determine if the last print scan had a print scan check. If it did, 46 more cycles must be allowed so that the character associated with the print scan check can be re-processed. In this event, CTR46 is set to 46 and the routine exits to ILS01. With each character emitter interrupt, CTR46, if non-zero, is decremented by one and routine exits to ILS01. When CTR46 is zero and no print scan error is detected, routine will branch to FC70 to check CTR48.
2. Check CTR48. CTR48 is set to 48 during call processing for alphameric printing. It is decremented by 1 for each successful print scan. If CTR48 is not zero, the routine exits to the EMIT routine to scan the I/O area for the next character to be printed. If the printing is numeric, CTR48 will be set to 22 by the EMIT routine (22 numeric and special characters). If CTR48 is zero, indicating that all printing is completed, the scan field is cleared and a 1 bit is placed in bit position 15 of word 8 (Core location 39) of the scan field.
3. Check CTR16. CTR16 is set to 48 during call processing time. It is used to count 16 of the 18 idle cycles required to complete a print operation. The counter is decremented by 3 for each of the 16 idle cycles. When CTR16 goes to zero, the SPSK counter is checked. If a space or skip operation is to be executed, the SPSK will contain the skip to channel number or a digit representing the number of spaces remaining to be taken. A space or skip command will be executed, the ISS counter will be incremented by 1, and the routine will exit to ILS01.

   If the value of the SPSK counter is zero, the control parameter of the print function is checked to determine if a space after print should be executed.
4. CTR2. CTR2 is set to twelve during call processing. It is decremented by 6 each time an idle cycle interrupt occurs. When CTR2 goes to zero, a stop printer operation is executed. The ISS counter is decremented by 1 and the routine exits to ILS01. A skip or space operation can be in progress at this time.

Errors: The PASS indicator is checked after printing (when CTR16 reaches zero). If it indicates that channel 9 has been passed, the routine exits to the user's error routine with a 3 in the Accumulator. If channel 12 has been passed, the user's error routine is entered and a 4 is placed in the Accumulator. Upon returning from the user's routine, the operation will continue as follows:

If the value of the Accumulator has been set to zero, no skip to channel 1 will be initiated. User requested carriage operations will be serviced as usual.

If the value of the Accumulator is not zero, a skip to 1 will occur.

Emit: A Read command will be executed to determine which typewheel character is to be printed next. Each character in the I/O area is checked and a bit is set in the corresponding position of the scan field for each matching character.

If the print function is Print Numeric, CTR48 is set to 22 and printing is suspended until the first numeric character is in position to print. When the I/O area has been scanned and the scan field set up, the routine exits to OUT.

DISK SUBROUTINES (DISK1)

Call Processing

- Determines type of function
- Checks status of routine and device
- Initiates requested operation

Operation: When entered via an LIBF statement, the contents of XR1, XR2, the Accumulator and Extension, and the status indicators are saved. XR1 is set to point to LIBF+2.

If a Test function is specified, the routine busy indicator is checked and the saved registers are restored. If the routine is busy, the subroutine exits to LIBF+3, if not busy, to LIBF+4.

If the function is not Test, store the address of the I/O area into IREAD and IWRITE. This data is saved and also becomes the first word of the read and write IOCC. Store the original sector address at SAVE2.* The effective sector address is computed by adding the file protect address to the original sector address if the displacement option is specified in the control parameter. If this address is over +1599, an error exit is made to location 41. Add eight to the sector address for each defective cylinder prior to and including the present cylinder (maximum three cylinders per disk). Word two of the IOCCs for Read, Write, Control, and Sense is now built. The ISS counter is incremented by one, the Retry Count is set to ten, and the Busy and First Count indicators are set. The usable word count (word count +1 to indicate the sector ID word) and the usable sector address are stored in the I/O area. The function is now initiated.

Seek: If a seek option is specified, set the control IOCC to indicate a cylinder count of one, modify CYLIN (internal cylinder count) by eight, and set the interrupt return to terminate the function. Execute a seek to the next cylinder.

If a seek option is not specified, modify the read IOCC to allow the sector address from the desired sector to be read. Call SBRTA (subroutine A) to initiate a seek to the cylinder specified by the sector address in the I/O area. Terminate the operation when the correct sector address is read.

Read (GET): Call SBRTA to determine if the read heads are at the correct cylinder. If they are not, SBRTA seeks the correct cylinder and reads the desired sector.

Write (PUT): The following paragraphs describe the write operations with or without a read back check.

Determine if the cylinder is file protected. If it is, terminate the function and exit to location 41.

If the cylinder is not file protected, call SBRTA to determine if the R/W heads are at the correct location. SBRTA seeks if required. When the correct sector has been located, write the data, terminate the function and exit to ILS02.

If a Read Back Check (RBC) is specified, modify the read IOCC to indicate an RBC with an I/O area address the same as the write IOCC. The program branches to SBRTA to execute the Read Back Check operation.

Write Immediate: Write disk data and terminate the function. No data error checks are made.

When the operation is terminated, the ISS counter is decremented by one and the Routine Busy indicator is cleared.

Errors: The subroutine will exit to the user's error routine if after ten tries the desired sector address is not located, an irrecoverable read error is found, or the read back check indicates an inability to write correctly.

Subroutine A (SBRTA)

- Check sector address read to determine physical location of heads
- Seek the requested cylinder
- Read the specified sector
- Process errors

Operation: Initially the internal cylinder location (CYLIN) is compared to the cylinder address specified in the sector address (TRAC) to determine if the R/W heads are at the correct location. If CYLIN and TRACK agree, a read is initiated at SBA10 to confirm the R/W head location and the return is modified to SBA13. If the location is incorrect, a seek is initiated towards TRACK. The sign of the difference between CYLIN and TRACK determines the direction (if CYLIN is less, the sign is negative and the seek will be towards the center of the disk). If a seek is necessary, the return is modified to SBA10. At SBA10 a read is initiated and the return is modified to SBA13. The routine will continue to loop between SBA10 and SBA13 (reinitializing a seek on

each pass) in an attempt to locate the correct cylinder address. If the correct cylinder is not found after ten tries, the routine exits to the user's error routine. If the read at SBA10 indicates that the actual location agrees with TRACK, the cylinder number is stored in CYLIN and the routine exits to the return address stored at SBRTA (BSI return).

MULT (Read or Write Multiple Sectors)

- Read or write data on consecutive sectors contained on more than one cylinder
- Call RBCRT routine

Operation: The original word count and sector address is stored in the I/O area. The remaining word count (stored in WKWC table) is tested. If its value is zero or negative, indicating that all words have been read or written, RBCRT is called. If RBC is not requested, the routine exits back to MULT; otherwise, the routine exits to the return address stored at RBCRT (BSI return). If the word count test indicates that more data is to be processed,the retry counter is set to ten and one is added to the sector address. If the three low-order bits of the sector address go to zero (overflow), a seek to the next cylinder is required. 320 is then added to IWRITE (the address of the I/O area), the last two data words in the I/O area (IWRITE-1 and IWRITE-2) are saved and the new sector ID is inserted in the I/O area in place of the last data word (IWRITE-1). The new word count is now determined and its value is inserted into the I/O area in place of the next to the last data word (IWRITE-2). IWRITE is set to point to the new word count. One is added to the sector number in the write IOCC, the read IOCC (IREAD) is modified to indicate the same I/O area and sector address as the write IOCC, and the routine exits to the return address.

RBCRT (Read Back Check)

- Check all written data
- Return to MULT if RBC is not requested

Operation: The RBCRT routine is entered from MULT when a cylinder has been filled and a seek is necessary or when all of the data in the I/O area has been written. If an RBC is not requested (RBC not zero) the routine exits back to MULT. If a check is required, the constant table is searched to determine the I/O area to be checked. A read IOCC is built using data saved from the call processing routine. The interrupt return is modified to return to the RBCRT routine. When all written data has been checked, the routine exits back to MULT.

FLIPPER ROUTINES (FLIP0, FLIP1)

- Read LOCALs into storage and transfer control to the LOCAL.
- FLIP0 is used with DISK0 and DISKZ.
- FLIP1 is used with DISK1 and DISKN.

Description

The FLIP routine contains two entry points. One is used with two-word calls (CALL), and the other with one-word calls (LIBF). After entry, the FLIP routine fetches the XEQ address and the LOCALs sector address in Working Storage from the Flipper Table (see Section 2, Supervisor, for a description of the Flipper Table). A test is then made to determine whether the LOCAL is currently in storage by comparing the sector address from the Flipper Table with the sector address of the last LOCAL read. If the LOCAL is not in storage, the word count is fetched from the Flipper Table.

The manner in which a LOCAL is read differentiates FLIP0 from FLIP1. FLIP0 reads a LOCAL into storage one sector at a time, whereas FLIP1 passes the total word count to DISK1 or DISKN and that routine reads in the entire LOCAL. After the LOCAL is read, or if it was already in storage, a check is made to determine if the LOCAL is a CALL or a LIBF. If the routine is a CALL, the return address is stored in the XEQ address of the routine and control is transferred to XEQ+1. If the routine is a LIBF, control is transferred to the XEQ address as specified in the Flipper Table.

FORTRAN I/O

The FORTRAN I/O routines are a part of Subroutine Library. These routines provide a link between the FORTRAN object program and the I/O devices.

There are two versions of FORTRAN I/O - SFIO and SDFIO.

- SFIO services non-disk I/O device; it supports standard and extended precision.
- SDFIO services the disk I/O statements; it supports standard and extended precision.

Each of these versions has ten separate entry points. The entry mnemonics and major functions are:

SFIO		SDFIO
SFIO	- performs I/O initialization	SDFIO
SRED	- accomplishes a READ operation	SDRED
SWRT	- accomplishes a WRITE operation	SDWRT
SCOMP	- completes WRITE operation (if necessary)	SDCOM
SIOI	- handles an integer, non-subscripted element	SDI
SIOIX	- handles an integer, subscripted element	SDIX
SIOF	- handles a real, non-subscripted element	SDF
SIOFX	- handles a real, subscripted element	SDFX
SIOAI	- handles an integer, non-subscripted array	SDAI
SIOAF	- handles a real, non-subscripted array	SDAF

Device Routines

There are 4 versions of the various device routines which perform the I/O functions. The version is denoted by the suffix added to the routine name. The suffixes are 0, 1, N, and Z. The suffix 0, 1, and N routines are general utility I/O routines. The suffix Z indicates a routine specifically designed to support FORTRAN programs.

See the publication IBM 1130 Subroutine Library (Form C26-5929) for a detailed discussion of these various routines.

Input Specifications

The object time I/O requirements of a FORTRAN program are indicated in the FORTRAN control records which precede the source program at compile time. These control records are interpreted by the FORTRAN Compiler (see Phase 1 in Section 5: FORTRAN) and during compilation the calls and parameters needed to accomplish the I/O functions are inserted into the object program under a 'LIBF *FIO' (see Phase 2 in Section 5: FORTRAN).

FIO Call

The calls and parameters inserted by the FORTRAN Compiler consist of a FORTRAN I/O initialization sequence and a table of calls (LIBF SFIO only) to the routines servicing the devices specified in the *IOCS control record.

Non-Disk		Disk	
LIBF	SFIO	LIBF	SDFIO
DC	TS	DC	S
DC	B		

The first parameter following the 'LIBF SFIO' denotes the trace device (T) if tracing is specified by an *TRANSFER TRACE or by an *ARITHMETIC TRACE control record and integer size and precision (S). S equals 4 or 6 if one-word integers are specified by the *ONE WORD INTEGERS control records. S equals 7 or 5 if either extended or standard precision is specified by the *EXTENDED PRECISION control record or by the default condition, respectively.

The second parameter contains the displacement (D), in words, to the next executable statement (the following I/O calls comprise a table rather than in-line executed instructions).

The SFIO entry to the non-disk I/O routine sets up the addresses and parameters which are needed by all the other entry points.

At object-time the call to SFIO must be executed prior to any other I/O call (in fact the 'LIBF SFIO', or SDFIO if present, is made the first executable statement in the object program by the FORTRAN Compiler). If a call to SFIO or SDFIO is not executed or is preceded by a call to some other I/O entry point, the program will WAIT with /F001 (non-disk) or /F103 (disk) displayed in the accumulator. Pressing PROGRAM START will cause control to return to the Monitor.

At the execution of a call to the SRED or SWRT entry point following execution of the call to SFIO, the call corresponding to the device specified in the SRED or SWRT call is located in the table of device routines. The SRED or SWRT routine then executes the call selected from the table. The called device performs its I/O function and returns to the SRED or SWRT routine.

1130 FORTRAN Non-disk I/O

The 1130 FORTRAN non-disk I/O services a non-disk device. The Z suffix device routines are used to perform the I/O functions.

Assuming that all devices were specified in the *IOCS control record, no tracing was specified, and extended precision was not specified, the I/O initialization sequence and calling table for an 1130 program appear as follows:

```
LIBF    SFIO
DC      0005
DC      12
```

```
LIBF    WRTYZ     (Typewriter)
DC      0
LIBF    CARDZ     (Card Read/Punch)
DC      0
LIBF    PRNTZ     (1132 Printer)
DC      0
LIBF    PAPTZ     (Paper Tape Read/Punch)
DC      0
DC      0
DC      0
LIBF    TYPEZ     (Keyboard)
DC      0
```

The position and order of the device routine calls are fixed; the calls are always separated by the 'DC 0'. If a device is not specified in the *IOCS control record, its corresponding call in the table is replaced by a 'DC 0'.

Summary of the 1130 FORTRAN Non-disk I/O

The following descriptions summarize the entry points and FORMAT scan routines which comprise the 1130 FORTRAN non-disk I/O (see Chart FA).

The ten entry points to the I/O routine are:

SFIO

- Saves the address of the I/O call table.
- Sets the integer length and trace device parameters.

SRED/SWRT

- Sets the Read/Write indicator.
- Sets the address of the FORMAT statement.
- Sets the I/O unit number.
- Sets the I/O call from the call table.
- Reads a record and goes to the FORMAT scan or clears the I/O buffer and goes to the FORMAT scan.

SIOF/SIOI

- Sets real/integer type indicator.
- Sets array element counter to 1.
- Sets the address of the variable.
- Goes to the FORMAT scan.

SIOFX/SIOIX

- Sets real/integer type indicator.
- Sets array element counter to 1.
- Calculates address of this element.
- Goes to the FORMAT scan.

SIOAF/SIOAI

- Sets real/integer type indicator.
- Places the number of elements into array element counter.
- Sets address of the first element of the array.
- Goes to the FORMAT scan.

SCOMP

- Checks REDO indicator.
- Outputs the I/O buffer, if ON.
- Checks the last format type for slash (/), if OFF.
- Returns to the calling program, if slash found.
- Outputs the I/O buffer, if slash not found.

The FORMAT scan:

FRMTS (FORMAT Scan)

- Checks the REDO indicator; executes I/O, if on; gets next word from FORMAT, if off.
- Checks for types E, F, I, and A; if one of these, checks that all items in the last list entry were processed; if so, returns to the calling program.
- Checks for types E and F, if all previous list items not processed; if one of these, saves decimal specifications.
- Saves field width specifications.
- Checks for E, F, I, A, X, and H types; checks for full I/O buffer; displays error code and WAITs, if full.

SLASH

- Sets REDO indicator on.

- Moves the FORMAT pointer one position.
- Sets the buffer pointer to the beginning of the buffer.
- Goes to the FORMAT scan.

RDO

- Sets REDO indicator on.
- Sets the FORMAT pointer to the beginning of the FORMAT statement.
- Determines if the array is exhausted; returns to the calling program, if yes; goes to the FORMAT scan if no.

GRPRP (Group Repeat)

- Moves the FORMAT pointer to the backspace number.
- Increments the repeat counter.
- Checks for completed repeat; goes to the FORMAT scan, if yes; backspaces the FORMAT pointer and goes to the FORMAT scan, if no.

FLDRP (Field Repeat)

- Increments the repeat counter.
- Checks for completed repeat; moves to the next position of the FORMAT statement and goes to the FORMAT scan, if yes; backspaces the FORMAT pointer and goes to the FORMAT scan if no.

TYPX (X-type Format)

- Increments the buffer pointer by positions specified in X-type.
- Moves the FORMAT pointer one position.
- Goes to the FORMAT scan.

TYPH (H-type Format)

- Sets the address of H data.
- Transfers EBCDIC characters - from storage to I/O buffer if a Write, from I/O buffer to storage if a Read - until all characters are processed.
- Checks for the H-type; moves the FORMAT pointer one position and goes to the FORMAT scan if yes; decrements the array count and address of array storage and goes to the FORMAT scan if no.

TYPIF (E, F, and I-type Formats)

- Initializes the parameters following a 'LIBF FST0' (floating store) or 'LIBF FLD' (floating load).
- Goes to Decimal-to-Binary conversion routine (DEC2) if a Read operation.
- Determines if storage mode is Real; loads Floating Accumulator if yes; loads accumulator and converts the fixed point value to a floating point value if no.
- Goes to the Decimal-to-Binary conversion routine (DEC2).

TYPA (A-type Format)

- Calculates maximum number of characters per variable.
- Gets storage address of A-type I/O area.
- Truncate left-hand characters if A-type data exceeds the maximum specification; insert blanks if data fails to fill the input area.
- Transfers EBCDIC characters - from storage to I/O buffer if a Write, from I/O buffer to storage, if a Read - until all characters are processed.

DEC2 (Decimal-to-Binary Conversion)

- Gets a character from the I/O buffer and increments the buffer pointer.
- Determines if the character is one of the valid subset; displays an error code and WAITs if not.
- Checks for a numeric digit; checks for an E (exponent) indicator; builds the binary exponent if E found; builds a binary mantissa if E not found.
- Checks for a decimal point; increments count of a decimal digits if decimal found; checks for remaining characters.

- Checks for duplication of the E (exponent) indicator, the sign, and the decimal point; checks for imbedded blanks; displays an error code and WAITs on any of these conditions.

DEC 50 +5 (Process Binary Built Mantissa)

- Checks if the mantissa equals 0; stores zeros to the the Floating Accumulator if yes.
- Adjusts for the exponent; normalizes the number.
- Checks if the mantissa sign is negative; stores the number in negative form in the Floating Accumulator if yes.
- Checks if the exponent is greater than 250 or less than 0, stores the number in the Floating Accumulator if not; displays an error code and WAITs if yes.
- Checks if the mode of data is Real; stores the Floating Accumulator at the storage address if yes, truncates the number in the Floating Accumulator at the decimal point and stores the resultant integer at the storage address if not.
- Decrements the storage address and array counter.
- Increments the FORMAT pointer and goes to the FORMAT scan.

BINRY (Binary-to-Decimal Conversion)

- Calculates the working buffer size.
- Stores the Floating Accumulator in the work area.
- Checks for sign; initializes to process positive or negative number.
- Checks for zero number; zeros out the exponent if yes.
- Checks if the exponent is greater than or less than 128; normalizes the binary mantissa, forcing the exponent to the base 128.
- Checks for an E-type format; sets the exponent sign and converts the exponent to 2 decimal digits if yes.
- Compares the data length with the field width; loads the buffer with asterisks if data overflows; loads the number with leading blanks, if necessary, if data does not overflow.
- Outputs mantissa sign and rounded number if I-type format.
- Outputs integer portion of the number.
- Outputs decimal point and fractional portion also if not I-type format.
- Outputs 'E', sign, and exponent if E-type format.
- Decrements the array count, increments the FORMAT.pointer, and goes to the FORMAT scan.

1130 FORTRAN Disk I/O

With the exception of SDFIO, SDRED and SDWRT, all calls to the FORTRAN disk I/O entries are identical to their counterparts in the FORTRAN non-disk I/O routines. The exceptions are:

SDFIO

The calling sequence is:

```
LIBF   SDFIO
DC     S
```

where S is a parameter specifying integer size and precision. Through SDFIO initialization, this parameter governs the number of words per element to be moved via the real and integer entries.

SDRED and SDWRT

The calling sequence is:

```
LIBF   SDRED   (or SDWRT for WRITE
                operation)
DC     FILE
DC     REC
```

where FILE is the ID of the disk file to be read or written (the file must have been defined by a DEFINE FILE statement) and REC is the record number within the specified file to be read or written. FILE and REC actually contain the addresses of the locations containing the required information.

All disk data is stored in binary; therefore, knowledge of the type of variable (integer or real) is necessary only to determine the number of words to be moved.

Summary of the 1130 FORTRAN Disk I/O

The following descriptions summarize the entry points and other major areas within the FORTRAN disk I/O routines.

SDFIO

- Checks and initializes integer precision.
- Checks and initializes standard or extended precision for real variables.
- Sets indicator showing SDFIO has been called.

SDRED

- Sets read indicator.
- Enters SRED1.

SDWRT

- Sets write indicator.
- Goes to SRED1.

SDCOM

- Sets exit to the calling program.
- Goes to DUFIP to perform final disk write.

SDF

- Sets real variable flag.
- Picks up word count for real variable.

SDFX

- Sets real variable flag.
- Picks up subscript from calling sequence.
- Obtains word count for real variable.

SDI

- Sets integer flag.
- Obtains integer size.

SDIX

- Sets integer flag.
- Gets subscript from calling sequence.
- Fetches word count for integer variable.

SDAF

- Sets real variable flag.
- Obtains number of elements to be moved from calling sequence.
- Picks up word count for each element.

SDAI

- Sets integer variable flag.
- Gets number of elements to be moved from calling sequence.
- Gets integer size.

SRED1

- Verifies that SDFIO has been called.
- Initializes total number of files for this job.
- Searches file table for specified file.
- Issues error condition if file not found.

DFND

- Gets starting sector address of specified file.
- Checks for valid record number.
- Calculates sector containing specified record.

RECOK

- Sets up associated variable.
- Calculates position in sector of the record.

DIO

- Clears REDO switch.

- Sets forced read for initial call to disk (if call is a WRITE, a read must first be given to preserve data that should not be altered.

DUFIP

- Performs call to disk I/O routine.
- Sets proper function parameter for next call.
- Checks REDO.
- If ON, returns to do a read.
- If OFF, checks count, if zero - exit, if not zero - go to MOVE.

MOVE

- Checks for full buffer, if yes set REDO and go to DUFIP.
- Checks for full record, if yes, move position pointer to next record, increment associated variable by one, and return to beginning of MOVE.
- Sets up move of variable, one word at a time, until variable is moved. If READ, move from buffer to LIST area. If WRITE, move from LIST area to buffer.
- Reduces variable count, when zero, exit.

The primary function of the System Loader/Editor is to load a disk pack with the programs and subroutines necessary to build a working Monitor System which is based upon the user's hardware configuration and individual requirements.

Initial System Load

During an initial load operation, the System Loader/Editor controls placement upon the disk of the Supervisor, DUP, the FORTRAN Compiler, the Assembler program, and the Subroutine Library.

During system load, a set of user-supplied control records causes deletion (bypassing) of programs and/or subroutines not desired or not usable by the system.

By means of the Load Mode control record the user can request that the Assembler and/or FORTRAN not be loaded. Accordingly the location of buffer areas, tables, and subroutine storage on the disk is shifted so as to provide as much Working Storage as possible.

The System Configuration Records specify to the System Loader/Editor (1) the devices present in the System and (2) the Interrupt Level Subroutines (ILS) and Interrupt Service Subroutines (ISS) to be included in the system pertaining to the specified devices.

As the storage locations of the FORTRAN Compiler and the Assembler program (if loaded) are established, the System Loader/Editor initializes the Communications Area (COMMA) with their starting addresses. The location of the Core Image Buffer (CIB), Location Equivalence Table (LET), User Area (UA), and other information contained in COMMA is also initialized by the System Loader/Editor.

After the system programs are on disk, control is transferred to DUP to load the subroutines and update COMMA and LET.

System Reload

In the event that the user wishes to reload the Monitor System programs (not including the Subroutine Library), the System Loader/Editor can perform a reload operation. This is controlled by a control record (Load Mode control record) inserted by the user in the System Loader/Editor following the last record of Phase E1 and preceding the first record of Phase E2. Bits 14 and 15 of the first 16-bit binary word of the Load Mode control record must call for exactly the same Monitor System programs to be stored as were present just prior to the initiation of the reload operation. During a reload operation, only the Supervisor (excepting COMMA), DUP, the Assembler program, and the FORTRAN Compiler are replaced upon the disk. If the Assembler program and/or the FORTRAN Compiler were not loaded during initial loading, they cannot be loaded during a reload of the Monitor system. Also, if the Assembler program and/or the FORTRAN Compiler were deleted by DUP after the initial loading, they cannot be loaded during a reload of the Monitor System.

SYSTEM LOADER/EDITOR INPUT

The 1130 Monitor System is available in either card or paper tape versions, ready for loading by the System Loader/Editor. The card version is supplied in a continuous card deck (see Figure 17), whereas the paper tape version is supplied in nine separate paper tapes which comprise the system (see below, Paper Tape Input).

User-Supplied Input

The user supplies to the System Loader/Editor the Load Mode control record and the System Configuration records. These records are generated by the user when the input is in card form. When the input is in paper tape form, the user simply selects the proper Load Mode and System Configuration tapes from the IBM-supplied set (see below, Paper Tape Input).

The Load Mode control record indicates to the System Loader/Editor whether an initial system load or a system reload is to be performed. This record also carries the indicators to the System Loader/Editor which cause bypassing of the Assembler program and/or the FORTRAN Compiler during system loading. This record is inserted between Phases E1 and E2 of the System Loader/Editor (see Figure 17).

The System Configuration records (SCON, REQ, and TERM) indicate to the System Loader/Editor the devices that are present in the system and their associated interrupt Branch addresses. Using these indicators, the System Loader/Editor loads only those Interrupt Level Subroutines (ILS) and Interrupt Service Subroutines (ISS), that are required in the system. These records are inserted immediately following Phase E2 of the System Loader/Editor (see Figure 17).

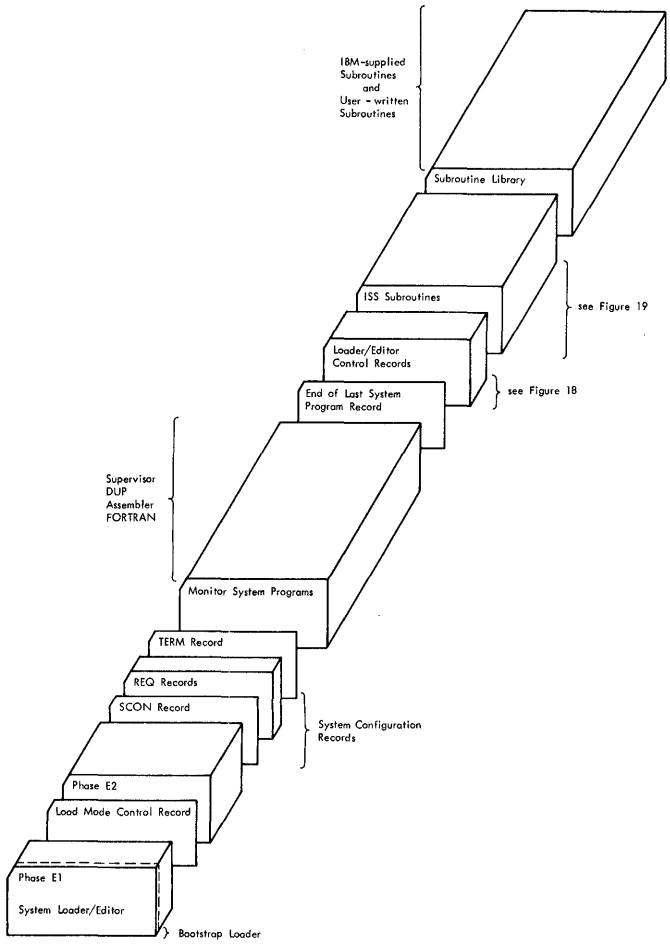


Figure 17. Organization of System Loader /Editor Input

See the publication IBM 1130 Monitor System Reference Manual (Form C26-3750) for a detailed description of the format and use of these user-supplied records.

IBM-Supplied Input

The following is a list of the IBM-supplied programs and control records which comprise the System Loader/Editor, excepting the User-supplied input (see Figures 17, 18, and 19).

1. The bootstrap loader and Phase E1.
   The bootstrap loader is comprised of the first six cards of Phase E1 in card form. The initial records of Phase E1 in paper tape form are the same bootstrap loader.

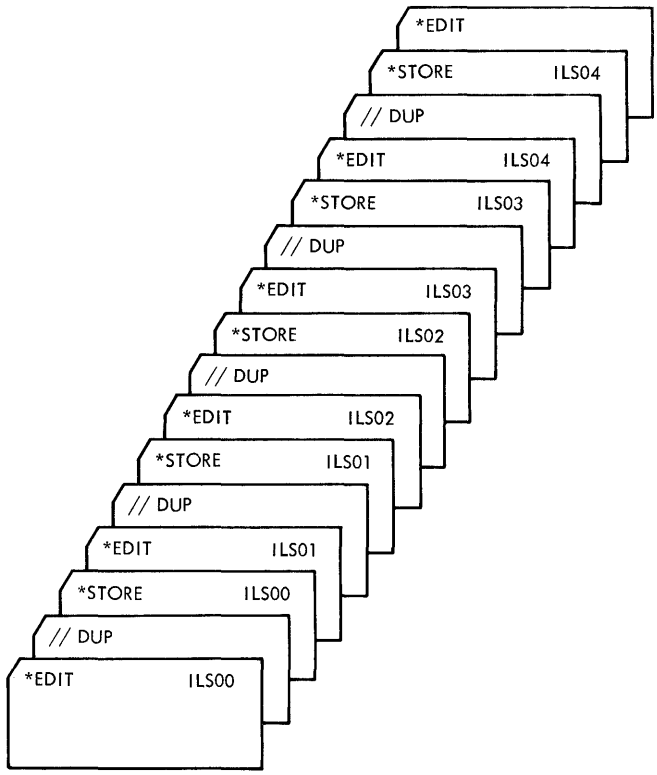


Figure 18. Loader /Editor Control Records

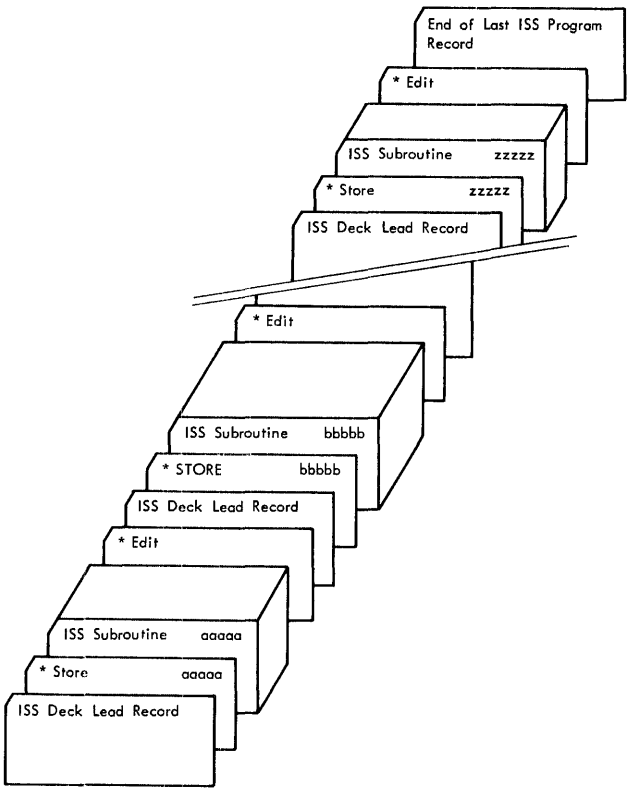


Figure 19. ISS Subroutines

2. Phase E2.
3. The Monitor System programs: the Supervisor (including COMMA), DUP, the Assembler program, and the FORTRAN Compiler.
4. The Loader/Editor control records.
5. The ISS subroutines.
6. The Subroutine Library.
7. The Load Mode control records and System Configuration records for paper tape systems (see below, Paper Tape Input).

Within the System Loader/Editor input, the following control records are found. They are listed by type code. The leftmost eight bits of binary word three of each control record contain the type code.

Type Code

01 Sector Break Record (absolute sector address)
This record is found within the Monitor System programs. It instructs the System Loader/Editor to load the following phase or routine at a sector boundary.

Format (in 16-bit binary code)

Word 1 - non-zero indicates that the phase to follow involves either a print device or an I/O device
bit 0 off = I/O device involved (paper tape or card)
bit 0 on = print device involved (Console Printer or 1132 Printer)

valid codes for word 1 include:
0001 = 1442 card read/punch involved in phase
0002 = paper tape I/O involved in phase
8001 = Console Printer involved in phase
8002 = 1132 Printer involved in phase

Word 2 - unused
Word 3 - type code; hexadecimal 0100, 0200, or 0900
Words 4-60 - unused

02 - Sector Break record (sector address relative to the last sector with data)
04 - *STORE DUP control record
08 - *EDIT special DUP control record
08 - // DUP Monitor control record
09 - Sector Break record (sector address relative to the last sector address established by a type 01 control record.
0A - Data (relocatable binary) record
0F - End-of-program (type F) record
10 - Load Mode control record (see User-supplied Input.)
12 - FORTRAN Compiler lead record
This record precedes the FORTRAN Compiler and indicates to the System Loader/Editor that the FORTRAN Compiler has been reached.

Format (in IBM card code)
Cols. 1-4 FORT
Cols. 5-72 unused

13 - ISS program lead record
This record carries the ISS number of the following ISS subroutine used by the System Loader/Editor in determining whether to load the following ISS subroutine.

Format (in 16-bit binary code)
Words 1-2 - unused
Word 3 - type code; hex 1300
Word 4 - ISS number (1,2,3,4,6, or 7) of following ISS
Words 5-60- unused

14 - TERM record (a System Configuration record; see User-supplied Input.)

20 - REQ record (a System Configuration record; see User-supplied Input.)

40 - Assembler program lead record
This record precedes the Assembler program and indicates to the System Loader/Editor that the Assembler program has been reached.

Format (in IBM card code)
Cols. 1-3 SAP
Cols. 4-72 unused

42 - Supervisor program lead record
This record precedes the Supervisor program (follows COMMA) and indicates to the System Loader/Editor that the Supervisor program has been reached.

Format (in IBM Card Code)
Cols. 1-4 SUPV
Cols. 5-72 unused

81 - End of last system program record

This record follows the last Monitor System program and precedes the Loader/Editor control records.

Format (in 16-bit binary code)
Words 1-2 - unused
Word 3 - type code; hex 8100
Words 4-60 unused

84 - SCON record (a System Configuration record; see User-supplied Input).

86 - End of last ISS program record
This record follows the *EDIT control record following the last ISS subroutine. It precedes the Subroutine Library programs. The System Loader/Editor does not analyze any ISS records other than the ISS program lead record (type 13) of each ISS subroutine and the End record (type 86) following the last ISS subroutine.

Format (in 16-bit binary code)
Words 1-2 - unused
Word 3 - type code; hex 8600
Words 4-60 - unused

Paper Tape Input

Paper tapes are provided by IBM which correspond to all the standard paper tape-disk system configurations and loading combinations. The appropriate System Configuration tape is to be selected by the user from the set furnished. The tape should be chosen which contains in its header a list of those devices which the user has in his system.

A Load Mode tape is to be selected to meet the user's requirements. During an initial System load, 7 to 9 tapes will be loaded depending upon whether the FORTRAN Compiler and/or the Assembler program tapes are included in the load.

Tapes 1 through 8 contain binary data and all records, except the bootstrap loader section of tape 1, are preceded by a word count. Tape 9 contains a combination of binary records with word count and control records. Each control record beginning with // b or * is preceded by a new line (NL) character and is ended by an NL character. The data between NL characters is in PTTC/8 code. No word count is present. In binary control records (those not beginning with //b or *) there are no NL characters.

Tape #	Contents
1	The bootstrap loader and Phase E1.
2	Load Mode control record (IBM supplies a separate tape for each possible Load Mode combination; the user must select for loading the number 2 tape that matches his desired load mode).
3	Phase E2.
4	System Configuration records (IBM supplies a separate tape for each possible configuration; the user must select for loading the number 4 tape that matches his desired configuration).
5	The Supervisor.
6	DUP.
7	The FORTRAN Compiler.
8	The Assembler program.
9	System Loader/Editor control records, the ISS subroutines, and the Subroutine Library.

GENERAL DESCRIPTION

Two phases comprise the System Loader/Editor, Phase E1 and Phase E2. On cards, Phase E1 can be identified by "E1" in columns 73-74 of the deck and Phase E2 by "E2" in columns 73-74. The program is organized to operate within 4095 words of core storage and will not use any location above 0FFF.

To successfully load the Monitor System, the user must supply a Load Mode control record and the System Configuration records (see above, User-supplied Input).

NOTE: The following descriptions of Phases E1 and E2 assume the input to be in card form. However, the processing for card input is identical to that for paper tape input, except as noted under Paper Tape System Loader.

Phase E1

Phase E1 of the System Loader/Editor is read into core by a 6-card bootstrap loader (RLB). The bootstrap loader occupies locations /0000 through /005D and /0E50 through /0F24. Phase E1 is loaded into locations /0028 to approximately /0A70. When execution of E1 begins, the first card to be read should be the type 10 (hexadecimal) Load Mode Control Card which the user has placed between decks E1 and E2. If the Load Mode card is missing, an error message will be displayed on the Console Printer and the program will stop.

If reload is specified, Phase E1 compares control record data with certain information in COMMA to determine if a reload is possible. In processing with either an initial load or a reload, Phase E2 is read and stored in sectors /0630 through /063A of disk Working Storage. The System Configuration records which follow E2 are read by E1 and stored in the E1/E2 Communications Area that will be used by both phases and updated as loading operations are executed.

When the SCON card is read, E1 clears the area into which REQ card data will be stored, and sets a switch (PAK) so that the REQ cards will be read into an 80 word card buffer without packing to 60 word format. As each REQ card is read, the program will branch to a routine (CONVT) which will convert the IBM card code to binary and store the result in table CFTA. Each card is stored next to and following the previous one in the table. More than six cards will result in an error message.

When the TERM card is read, E1 will search through table CFTA checking each number from each card for validity and at the same time will build another table (AUXT2) whose words correspond to Interrupt Branch addresses. The words which correspond to IBAs that will be used are set positive, while those that will not be used remain zero. When the tables are finished, they are written to disk sector /0632 and control branches to routine LDPT2 to load E2 to core. Nine sectors will be loaded to core and will occupy locations /03C2 to /0F02. Control will exit from E1 and enter E2 at location /0504 (INIT2).

Phase E2

At SET10 Interrupt Branch Addresses will be reset to link with the ILS routines in E2. At SETDT, defective disk track data (originally from DPIR) will be set in the DISK0 routine of E2. Two additional sectors (/0630 and /0631) will then be read from disk into core to complete the overlay. From the REQ card data assembled by E1, two words will be set up at locations /047C and /047D which will reflect the principle I/O and principle print devices, respectively. During the system program load, if data is present in word 1 of a sector break card, that data will be compared with word /047C or /047D and that phase will be bypassed if no match occurs.

If an initial load is to be performed (determined in LDMDD subroutine), reading of the system program decks will commence.

If a reload is to be performed, sector 8 (COMMA) will be read into the disk buffer labeled BUFFI. If the FLET entry in COMMA is not zero, it will be stored in the word called CIBA in E2. If FLET is zero, the CIB address from COMMA will be placed in CIBA. During loading of the system programs, an error will be displayed if any sector requested to be loaded is numbered as high or higher than the contents of CIBA.

The words in COMMA which indicate whether or not the FORTRAN Compiler is on the disk and whether or not the Assembler Program is on the disk will be checked and compared with the type of load requested by the Load Mode card. If the reload is to be different from the present status of the disk pack, an error message will be displayed.

If no error in the Load Mode card is detected, a switch (SPVSW) is set on (non-zero) which results in the System Loader/Editor passing all system program cards until the SUPV card is encountered. This will be located in the system deck beyond those data cards which comprise sector 8 (COMMA). Beyond the SUPV card, loading will progress until the FORT card is encountered. During either an initial load or reload, the Load Mode card (which has been saved at MODCD) is tested to see if the FORTRAN Compiler

should be loaded. If not, switch A (SWA) is set positive and all subsequent cards will be bypassed until the SAP header card is read which turns SWA off (to zero). In like manner, MODCD is tested to see if the Assembler program should be loaded. If not, all cards will be bypassed until the type 81 End of System programs card is reached.

At the time the SAP card is read, an indicator (SAPX) is set on. When the first data card of the Assembler program is processed, the SAPX indicator will cause a branch to subroutine ASMCK. If the FORTRAN Compiler was not loaded and the Assembler program is to be loaded, the beginning sector address will be forced to /0080 with the constant ADRF. Normally the FORTRAN Compiler starts at sector /0080 and the Assembler program at /00E8.

During card to disk loading of any system program, if a sector break card is read which contains data in binary word 1, that phase involves some kind of I/O device and is to be loaded only if the device represented in the break card is equivalent to the principle I/O device of the system in the case where the code in word 1 refers to I/O devices. If the code refers to a print device, the phase will be loaded if the print device indicated is the principle one of the system. Otherwise switch B (SWB) will be set on (non-zero) and cards will be passed until a sector break card is reached. SWB is then set off (to zero) and sector break processing starts again.

In cases where the system includes both card I/O and paper tape I/O, the principle I/O device is interpreted to be the card read/punch. If an 1132 Printer is present with the Console Printer, the 1132 Printer is interpreted to be the principle print device. This is reflected in COMMA (sector 8).

Each time that a sector break card is read, the data-card check-sum routine (CKSR) is reset so that the data cards following the sector break card may begin with any sequence number. All subsequent data cards must then be in sequence until the next sector break card or an error message will be displayed.

As the system programs are loaded to disk, the highest sector number used is saved so that the Core Image Buffer (CIB) can be positioned at the first available cylinder beyond the last sector used.

When the End of System programs card (type 81) is read, the System Loader/Editor will branch to ENDSY and test the load mode. If a reload is being performed, E2 will be cleared from disk Working Storage (WS) and operations will halt at 0FFF. No more records will be read. COMMA will remain as it was when the reload began.

If an initial load is being performed, control will branch to LDSKL to load the Skeleton Supervisor from disk to lower core. At MILS in the System Loader/Editor the data in table AUXT2 will be used to generate for COMMA a code representation of the devices (see discussion above of principle devices) present in the system. The two words required are placed in core locations /0060 (principle print) and /0061 (principle I/O). If the FORTRAN Compiler was loaded, the beginning sector address will be placed in location /004A and if the Assembler program was loaded, its beginning sector address will go to /004B. If no program was loaded, the corresponding locations will contain zero.

Core size of the CPU will be stored in /007E. The CIB address (described above) will be placed in /004D and 3 cylinders will be reserved following that address. The address of FLET is set to zero and placed in /004F. The sector address of LET in /0050 will be 3 cylinders greater than the sector address of CIB. The User Area (UA), Working Storage Base and Working Storage Adjusted sector addresses will be equal and one cylinder greater than the address of LET. This value will go to locations /0051, /0052, and /0053. The UA sector address is converted to a disk block address and stored in locations /0058 and /0059. /0005 is stored in locations /0056 and /0057. Before this data is established on the disk, the 320 word buffer (BUFFI) is cleared to zero and the fourth word following the sector address is initialized to /013B. This buffer is then written to disk at the LET address established.

The region in lower core starting at location /0028 and extending 320 words is written to sector 8 (COMMA) of disk at REST8. Control goes next to MFILS where the highest level ILS that the system will require is generated.

To determine what ILS is to be built first, and how many will be required in all, the AUXT2 table is entered and the first word containing a positive number will correspond to some IBA which in turn corresponds to some interrupt level. When such a word is found, it will be set negative to indicate that it should be ignored next time.

In core is a basic framework for single device ILS routines and a multiple ILS routine. If the configuration deck contained a device which is on level 0 or level 4, an indicator (SETMI) is set which will cause ILS04 to be the last ILS routine generated.

If the positive word found in table AUXT2 corresponds to an IBA less than 12, the disk format header of the single device ILS frame will be set up to match the ILS under construction. SLW13 will be loaded with the level number. The ILS name, in truncated EBCDIC, will be loaded at SLW11. Data required by the DSF and Core Image Loader will be entered at CILO (the first word of the ILS). The area code will be loaded to CILA. Disk System

format indicator words are included as constants at the appropriate places in the framework.

When the ILS is finished, sector 8 (COMMA) is read from disk so that the current Working Storage address may be determined. The ILS is then written to disk at that address. If this is the first ILS that has been built, the next card in the hopper will be *EDIT.....ILS00. When this card is read, control will branch to EDITC where an indicator (FILS) will be tested to see if the Supervisor has been called. If it has not, control will go to the Supervisor provided that the cards in the hopper match the ILS that has been built. This is determined by testing columns 24 and 25 of the *EDIT card. If they do not match the name of the manufactured ILS, cards will be passed until the System Loader/Editor stops at an *EDIT card whose name field does match. At this point the next card in the hopper will cause the Supervisor to call in DUP, which will in turn read the next card. The record which DUP reads will be of the type *STORE..... WS..UA..ILSXX where XX represents the correct name of the ILS in Working Storage.

After the Supervisor has been called once, indicator FILS will be set to zero and thereafter the System Loader/Editor will transfer control directly to DUP Control (DCTL) at /027C after positioning the correct *STORE card in the hopper for DUP to read. When each DUP Store operation is completed, DUP will read a special control card (*EDIT) which will cause it to reload the System Loader/Editor to core from location /03C4 upward. DUP will then send control to /03C4 where a long BSC will send control to the System Loader/Editor at NEWIL.

The System Loader/Editor will continue to search table AUXT2 until no more positive words can be found. The re-entry point from DUP will then be set to point to ISSDK since the next processing operations will involve ISS decks.

When the last required ILS has been stored, the System Loader/Editor will encounter a lead record (type 13) ahead of each ISS routine. The System Loader/Editor will compare the ISS number of the following ISS subroutine with the System Configuration data and will either bypass the ISS subroutine or call DUP to load it to the User Area. When the last ISS subroutine has been processed, a control record (type 86) will signal the System Loader/Editor to clear itself from disk Working Storage and transfer complete control to DUP to load all remaining subroutines.

1130 Paper Tape System Loader/Editor

A special paper tape reading routine is employed in place of CARD0 and some differences, to be explained, exist in the way the input buffer (B) is filled. Otherwise, all input record processing and program logic is no different from the card system. In the paper tape system all binary input data goes directly into buffer B rather than first being read into buffer A and then compressed into buffer B.

Upon entering the paper tape reading routine at PAP00, buffer B is cleared and bit 5 of the tape reader DSW is tested. If not ready, routine will wait at /0C35. If ready, the first frame will be read. At PAP60, an indicator (PAK) will be tested to determine if the record should be in binary or in PTTC/8 format. If binary, control will go to PAP62 where the first frame will be treated as a word count provided that it is not a blank or a delete code. If the first frame contained a word count that was not zero, negative, or over /003C, the quantity of words indicated will be read into buffer B (two frames per word) and control will branch to CARD1 for record processing.

PAK will be set to zero (binary) throughout loading of the system programs. When the type 81 record which precedes the Subroutine Library is read, PAK will be set to 1 to cause the next input record to be read in PTTC/8 mode. In such a case, control will branch to that section of the paper tape routine labelled PAP61. Since each record in PTTC/8 code begins and ends with a new line (NL) character, the first NL character will be ignored and the last NL character will be interpreted as the end of the record.

As each PTTC/8 character is read in, a table, whose ending address is TABLE , is searched to find the binary equivalent of the frame, had the control record been in binary instead of PTTC/8. Not all characters are included in the table because the System Loader/Editor requires only those parts of control records which would be found in columns 3, 4, 24, and 25 of the same records on cards instead of tape.

The data gathered is placed in buffer A (as if it had come from a binary card) and when the ending NL character is reached, control branches to a routine called STUFF which packs the data from buffer A to buffer B. The binary data in buffer B is then processed at CARD1. When the System Loader/Editor finishes the last ILS control record which it will read, PAK is set back to zero.

The next record is an ISS control record (type 13) in binary format. If it is determined from this record that the following ISS should be loaded, PAK will be left at zero and control will branch to DUP at /027C. If the ISS is to be bypassed, PAK will be set to 1 since the next control record will be in PTTC/8 code (*STORE.....PT..UA..XXXXX). After this record is read, PAK will be reset to zero and binary records will be read until the End of program (type F)

record is reached. At that time, PAK is reset to 1 so that the next control record (*EDIT.....) is read in PTTC/8 mode. PAK is then set to zero again in anticipation of a type 13 binary record.

After all of the ISS programs are processed, the System Loader/Editor will clear itself from the disk and turn over complete control to DUP to load the remainder of the Subroutine Library.

During a reload operation, the System Loader/Editor will expect to find the SUPV binary control record within the first twenty records following the TERM record. If it is not found, a tape has been placed on the reader out of sequence. Error message 3 will be displayed.

Core Allocation Summary

Figure 20 reflects core after the bootstrap loader (the first six records of Phase E1) is in core.

Figure 21 reflects core between the reading of type F record of Phase E1 and the reading of the last REQ record of the System Configuration records following Phase E2.

When the TERM record is read, Phase E2 is loaded from disk to core as in Figure 22.

After sector /063A has been read from disk, control is transferred to Phase E2 at location /0504 and sectors /0630 and /0631 are read to core as shown in Figure 23. During reload operations, the core map remains as in Figure 23. End of reloading occurs when the type 81, end of last system program, record is read.

During an initial load, the Skeleton Supervisor is read from disk to core after the type 81 record is read. Figure 24 reflects core until the System Loader/Editor branches to location /0038 to call the Supervisor.

At this time one of the ILS control records beginning with //b will be the next record to be read. The data from the System Configuration records is used to determine which of these records the Supervisor should read so that DUP in turn reads an *STORE record which contains the ILS name which matches the ILS routine that the System Loader/Editor has placed in Working Storage. (See Figure 1 in Section 2.)

After the Monitor control record (with //b DUP) is read, DUPCO is loaded, and control is transferred to it. DUP reads a STORE record and reads from disk to core whatever DUP phases are necessary to perform the STORE function. Figure 8 (Section 3) reflects core until DUP completes the STORE operation and reads the *EDIT record which follows each *STORE record in the ILS group. The *EDIT card causes DUP to load the System Loader/Editor from disk to core as in Figure 25 and to branch to location /03C4.

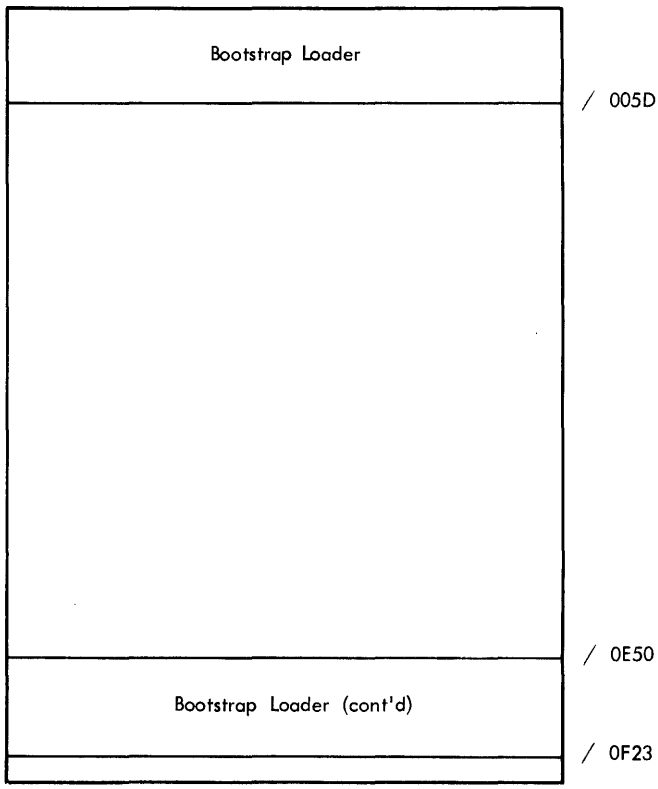


Figure 20. The Bootstrap Loader

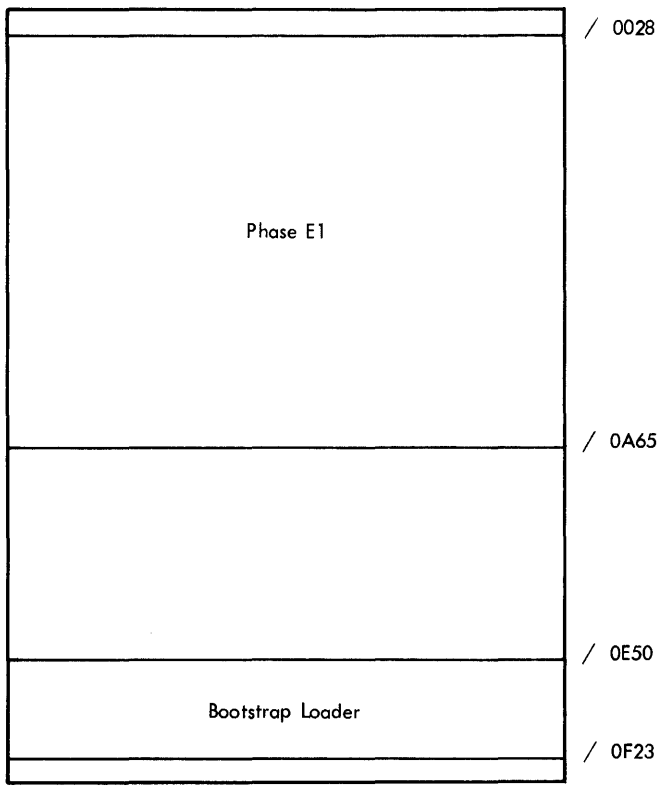


Figure 21. Phase E1

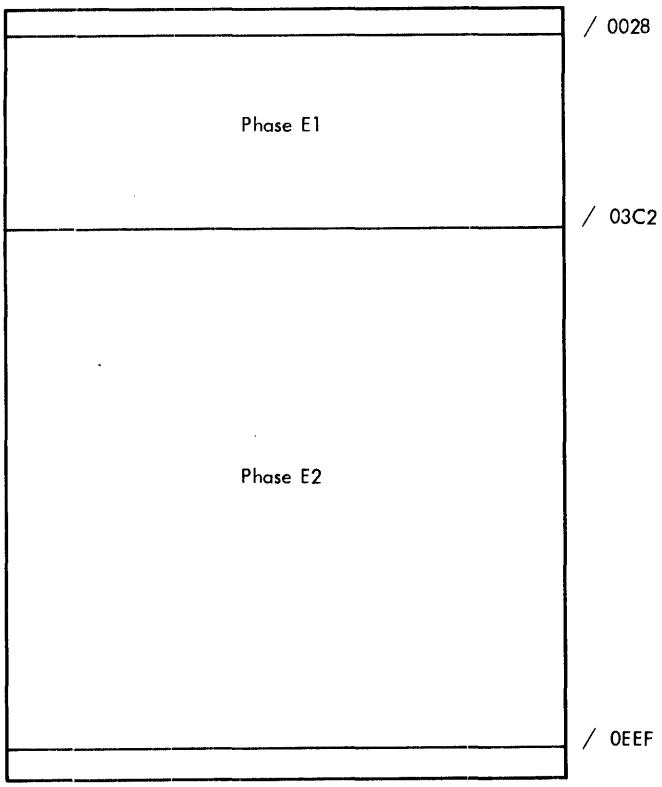


Figure 22. Phase E2 - Part I

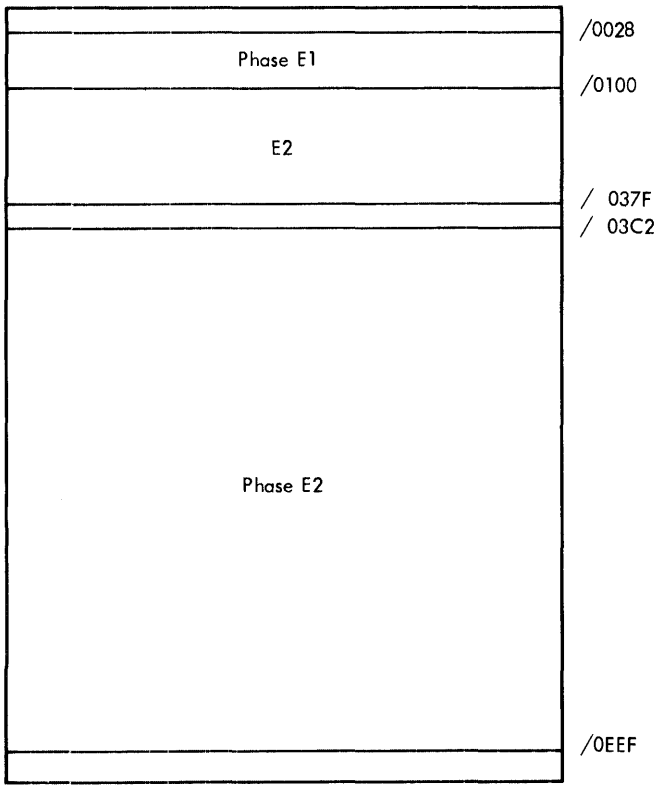


Figure 23. Phase E2 - Part II

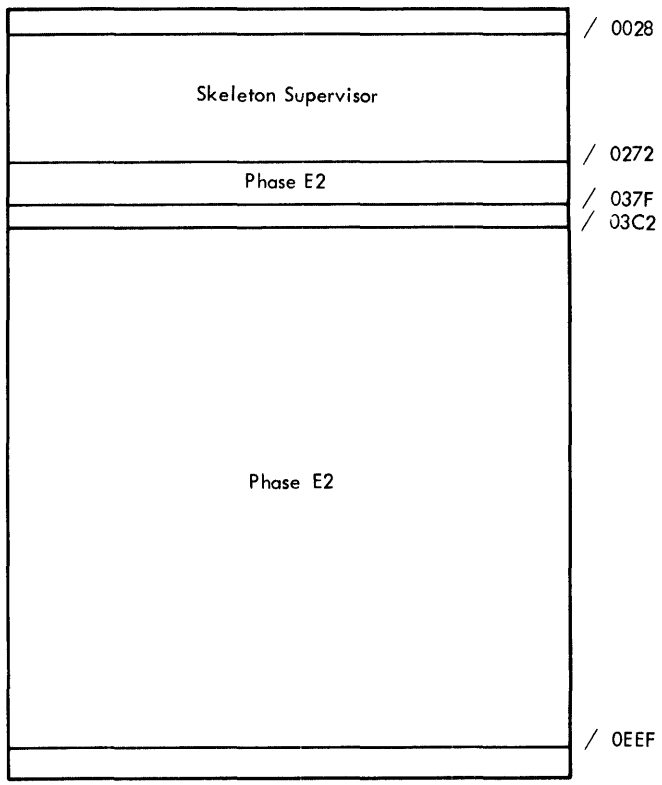


Figure 24. Phase E2 with the Skeleton Supervisor

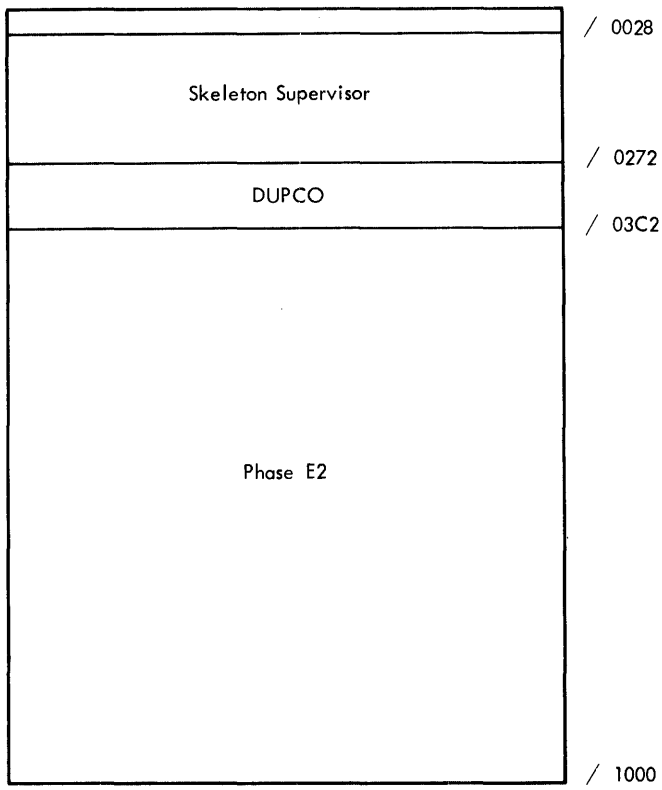


Figure 25. Phase E2 with DUPCO

When the System Loader/Editor again needs DUP to perform a STORE operation, records are passed as necessary by the System Loader/Editor until an *STORE record is the next record to be read. At this time, a long BSC is performed to location /027C and DCTL overlays core, as in Figure 8, with those phases necessary to store the ILS routine.

When the ILS routines have been stored, the System Loader/Editor (read into core by DUP at the end of each STORE function) tests for the type 13, ISS deck lead record. The System Loader/Editor remains in core (Figure 25), reading cards until an ISS subroutine is reached that should be put on the disk. At this time, a branch to DUP is executed and DUP is again brought into core (Figure 8).

At the end of the last ISS deck, a type 86, end of last ISS program record causes the System Loader/Editor to clear itself from the disk and branch to DUP. For the remainder of the subroutine storing operations, DUP is in complete control (Figure 8).

ROUTINE DESCRIPTIONS

This section describes the principal subroutines, entry points, and tables within the System Loader/Editor program. In addition, other subroutines and areas referenced by the program are described.

All references are in terms of labels/symbols used on both the flowcharts (Charts FB, FC, FD, and FE) and the program listings. To examine a specific portion of the program, the user can:

1. Find the area of interest on the logical level flowchart.
2. Use an identifying label/symbol from the flowchart as a cross-reference to both the entry point descriptions (contained in this section of the manual) and the program listing.

The program listing, with its descriptive notations or comments, provides a step-by-step analysis of all areas of the program.

E1/E2 Communication Area

During analysis of the System Configuration Records furnished by the user, Phase E1 sets up information in a region located at approximately /03C2 – /0502 in core. To preserve this information when core is overlaid by DUP, the area is written to disk storage before each DUP operation during ILS and ISS loading. Updating of the E1/E2 Communications Area on disk is done when necessary by the System Loader/Editor.

This area contains tables AB, AC, and CFTA, an image of the Load Mode control record, principal print and input/output device information, and various other indicators.

Primary Program Entry Points/Labels

Phase	Name/Label	Function and/or Description
E1 & E2	SET10	Initializes Interrupt Branch Addresses for the ILS routines which are part of the System Loader/Editor.
E1 & E2	START	Entrance to record read routine. On the card system, CARD0 is used.
E1	MODE	Performs checks on the Load Mode control record and sets up indicators in the E1/E2 Communications Area.
E1	SETDT	Picks up defective track information which Disk Pack Initialization Routine (DPIR) has written in sector zero, and stores it in both the E1 disk routine and in the E1/E2 Communications Area. An attempt is made to confirm that DPIR has been executed. If it definitely has not been executed, an error message is displayed and program will halt.
E2	ABSOR	One of several entry points in routine which loads system programs to disk. Sector addresses are established in one of three ways, depending upon the type of Sector Break control record processed. Ordinary data will start at the beginning of a sector and will progress until that sector is filled. The completed sector will then be written to disk and the next successive sector read down. Record information may overlay all or part of the sector being processed. This will continue until the program is loaded or until a

Phase	Name/Label	Function and/or Description
		Sector Break record is read and a new sector address is established.
E1	SCON	Clears area in which "AB" and "AC" auxiliary tables will be built and feeds control to routine labelled "CONVT" which will accept IBM card code and convert it to binary. The DCBIN conversion routine is used in this operation. Converted information from the REQ System Configuration records is then stored in CFTA tables of the E1/E2 Communications Area.
E1 & E2	LDPT2	Reads down sectors from disk as directed by three initialized control words: SCNUM is the address of the first disk sector to read from. PGLTH is the number of sectors to be read. HXCFG is 2 less than the core location to read into.
E2	INIT2	When E2 is loaded to core from disk, either by E1 or DUP, INIT2 will be branched to via the E1/E2 Communications Area. Interrupt branch locations in core will be set with the ILS routine addresses in E2 and control will go to LDMDD.
E2	LDMDD	Loading of system programs will begin in the case of an initial load. For reload, those records which could change COMMA are bypassed and the system load is started when a SUPV control record is reached. This control record immediately follows that part of the Supervisor which constitutes COMMA.
E2	IOCSD	After completion of the last ILS storing operation, the ISS Subroutines will be encountered. Each of these is preceded by a control record (type 13) which contains the ISS number common to the routine. If the number is not among the REQ card data, the routine will be bypassed since no device is present in the user's system which can handle it. If it is a usable routine, control will be transferred to the segment of DUP which remains in lower core. DUP will then store the ISS subroutine and reload the System Loader/Editor to core.
E2	CLEDT	A control record following the last ISS routine will signal the System Loader/Editor to clear itself from disk and transfer complete control to DUP. All remaining subroutines will then be loaded and final updating of COMMA will be under DUP control.

Additional Referenced Entry Points/Labels

Phase	Name/Label	Function and/or Description
E1 & E2	A	80-word buffer into which records are read by card or paper tape input routine.
E2	ABC	Switch testing routine entered at the time a Sector Break card is read. Its purpose is to determine if a phase should be bypassed or loaded.
E1 & E2	AC	Start of a table setup by E1 which contains IBA information.
E1 & E2	B	60-word buffer area that, if used, contains information

Phase	Name/Label	Function and/or Description
		from Buffer A in compressed format.
E1 & E2	BASE	Contains sector address established when last absolute Sector Break card was read.
E1 & E2	BUFFI	Address of the start of main buffer used in disk operations.
E1 & E2	B4HEX	Subroutine which will convert limited set of binary characters to EBCDIC code.
E1 & E2	CARD	Determines type of record read by testing word 3.
E1 & E2	CFACT	Correction factor used during system load which results in the first loadable data word following a Sector Break record being placed at the start of a sector.
E1 & E2	CFTA	Table where System Configuration records are stored after converting their contained data from IBM card code to binary.
E1 & E2	CFTAL	Number of complete records stored in CFTA table.
E1 & E2	CILA	Word in the single device ILS framework where the area code is placed.
E1 & E2	CILO	ILS information for the DSF and Core Image Loader.
E1 & E2	CKSR	Checksum routine. Controls are reset after each Sector Break card so that record sequence breaks can be tolerated between loaded phases.
E2	CLDUP	Sequence which will transfer control to DUP Control (DCTL).
E2	CZ	Routine to determine core size for COMMA using "wrap around" method.
E1 & E2	DSKAD	Word in which number of current sector to be written to is maintained as loading progresses.
E1 & E2	EDIT	E2 will be loaded from disk to the location equated to EDIT when required. DUP also contains this address.
E1 & E2	EOP	Branch point when a type F record is read. A file operation is initiated at this point so that the probably unfilled in-core buffer is written to disk.
E1 & E2	ERROR	When a disk error occurs, an indicator is set and control is returned to the disk routine to turn off the interrupt level. Upon exit from disk routine, the indicator is interrogated and if on, an error message is typed and program stops at a WAIT instruction. If PROGRAM START is pushed on the console, a retry of the same operation will be initiated.
E2	FORTT	Compares new and old (in COMMA) FORTRAN Compiler address during a reload operation.
E1 & E2	FORTX	Indicator which communicates from the Load Mode record to E2, indicating whether the FORTRAN Compiler is to be loaded.
E2	HIGH	Routine which saves the highest disk sector number loaded during system load. Later used to establish the CIB address and others in COMMA.
E2	IBANO	Word which indicates what interrupt branch address is being processed during ILS generation.
E1 & E2	ICNTR	Contains contents of word 1 from data records.
E2	INLET	Routine to set up the sector which will contain LET following proper initialization.
E1 & E2	ISW	Used as a "busy" indicator for the Console Printer routine.

Phase	Name/Label	Function and/or Description
E2	JADK	Used in preparation for calling DUP to establish the core location that sector 9 of the Skeleton Supervisor will be loaded to.
E2	LDSKL	Routine to load the Skeleton Supervisor to lower core. This includes sectors 8, 9 and A.
E1 & E2	MODCD	Image of the Load Mode control record.
E2	REST	Routine to restore updated table and the E1/E2 Communications Area to disk.
E2	REST8	Routine to write initialized COMMA (sector 8) to disk.
E2	SAPP	Compares new and old (in COMMA) Assembler program addresses during a reload operation.
E2	SAPX	Indicates to E2 whether the Load Mode control card requests that the Assembler program be loaded.
E1 & E2	SCTRI	Indicator set on at each Sector Break record; it causes following data record to receive special processing.
E2	SETMI	Indicator that will cause generation of ILSO4 routine if set on.
E2	SLW13	Word in single device ILS routines corresponding to the level number.
E2	SPPAS	Routine to pass all records except type 13.
E2	SPVSW	Switch to bypass COMMA region of Supervisor.
E1 & E2	DISKO	Routine to wait for completion of disk operation.
E1 & E2	WBSKT	Double word in loading routine which in leftmost 16 bits will contain indication of whether record information is to go to current sector. Rightmost 16 bits will contain relative word location within the sector.
E2	WD25A	Used to advance records, if necessary, to the point where DUP will receive a STORE record containing a name which matches the name of the ILS routine to be stored.
E2	WS	Routine to pick up current Working Storage address from COMMA before an ILS routine is written to disk for transfer by DUP to the User Area.

The 1130 Disk Monitor System Maintenance Program (IBM00) is the program by which the user updates the 1130 Disk Monitor System. The program IBM00 is a part of the IBM-supplied 1130 Disk Monitor System. With it, the user can update the Supervisor, Assembler, DUP, and FORTRAN Programs and the Subroutine Library as new releases are provided.

The input to the program can be in either card or paper tape form. However, the input must come from the principal I/O device.

Figure 26 shows the relative organization of core storage during execution of the maintenance program.

See Appendix F of the publication IBM 1130 Disk Monitor System Reference Manual (Form C26-3750) for an explanation of control record setup, header and data record formats, and operating procedures. Also see the same source for the description of the paper tape input, system messages, and error diagnostics.

The following descriptions summarize the routines which comprise the basic logic of the program IBM 00. The labels at the left correspond to the labels to be found in the program listings (see Charts FF, FG, and FH).

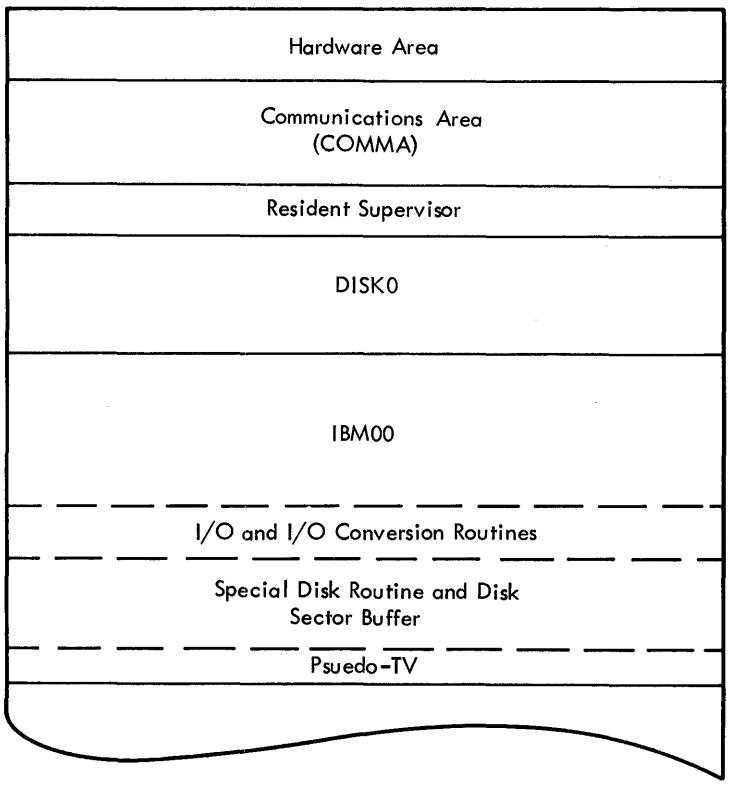


Figure 26. Storage Layout During Execution of the System Maintenance Program

START - Initializes the pseudo-TV and the IL subroutines; obtains from the Disk Communications Area (DCOM) the modification level and the version number and types them out; determines the principal I/O device for the system from COMMA; reads a record from that device.

PTSW1 - If the input is from paper tape, converts the input record from PTTC/8 to IBM card code.

CDRT1 - Determines if the input record is a valid System Program header record.

HRCVT - Extracts from the first header of a modification the total record count for that modification.

HDRWD - Converts to binary the word count, the relative word number, and the sector address obtained from the input header.

MLEV - Converts to binary the version number and modification level obtained from the input header.

S1SUB - Turns off the disk non-write switch, enabling the program to make modifications to the Monitor Programs.

HDCHK - Checks for the names FOR, ASM, SUP, and/or DUP in columns 1 through 3 of the System Program header record; continues accordingly.

SWFOR - Determines if FORTRAN is present as a Monitor Program; if not, turns on the disk non-write switch.

SWASM - Determines if the Assembler is present as a Monitor Program; if not, turns on the disk non-write switch.

VERSN - Checks the version number and modification level from the input header against those found in DCOM. If the new numbers are valid, the program proceeds; if not, an error message is printed.

ASMSW - Computes an absolute sector address for an Assembler modification from the relative sector address obtained from the input header.

SUBHR - Checks for a Subroutine header (the name SUB in columns 1 through 3 of the input header). If the name is found, the program continues; if not, the program initializes for a return to the Supervisor.

MOD1 - Warns the user that an attempt is being made to update the Monitor to modification level 1 and then WAITS. If PROGRAM START is pressed, the program continues.

RDISK+2 - Updates the modification level in DCOM after a modification has been successfully completed.

ENDOJ - Types the update complete message.

ENJ02 - Types the version number and new modification level; returns to the Supervisor.

CKSMG - Determines that the checksum contained in the input data record is correct. If incorrect, an error message is typed and the modification is terminated.

CSMOK - Moves the modification data from the input data record to the disk sector buffer. The word count from the data record and the relative word number and total word count from the header record are used to make the modification.

NWSW - Writes the modified sector onto the disk from the disk sector buffer when the modification is complete. This routine then reads the next input record. If the input is from paper tape, the PTTC/8 is converted to IBM card code.

If the disk non-write switch is on, this routine does not write the disk sector buffer onto the disk.

MCRCK - Determines if the input record is a Monitor control record. If it is, the program returns to the Supervisor; if it is not, the program terminates with an error.

A2P03 - Reads the sector to be modified from the disk into the disk sector buffer.

RDATA - Reads an input data record, either card or paper tape.

PCKNG - Packs 80 card columns into 60 binary words.

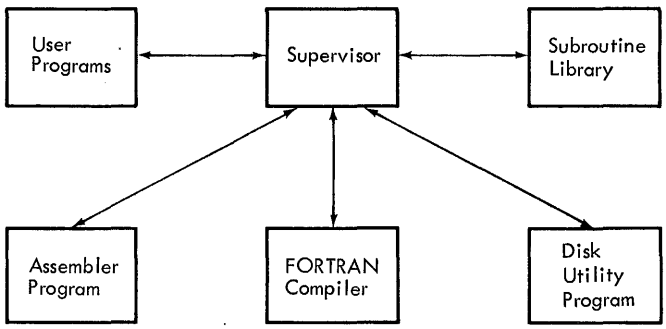


Chart AA. The 1130 Monitor System

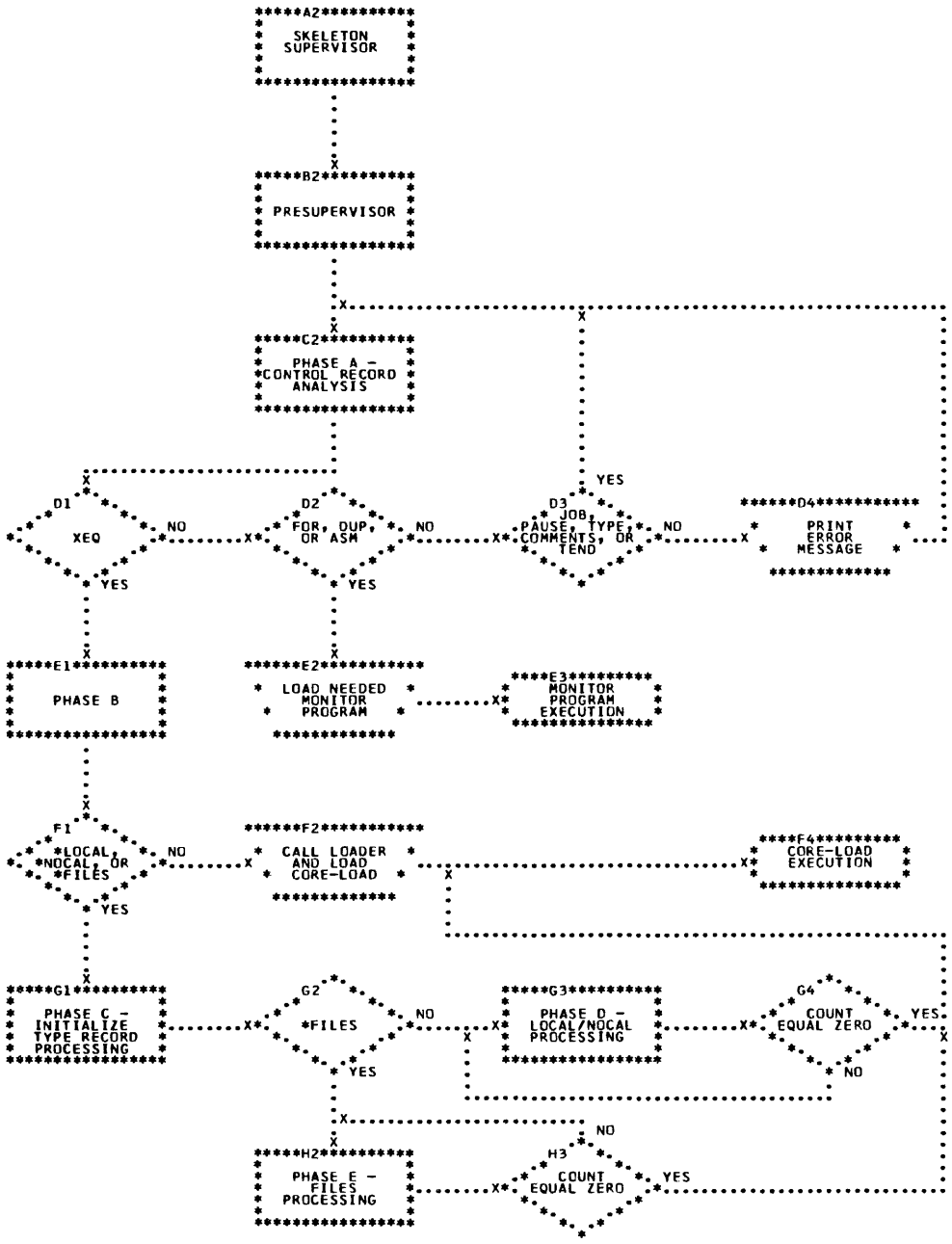


Chart AB. The Supervisor

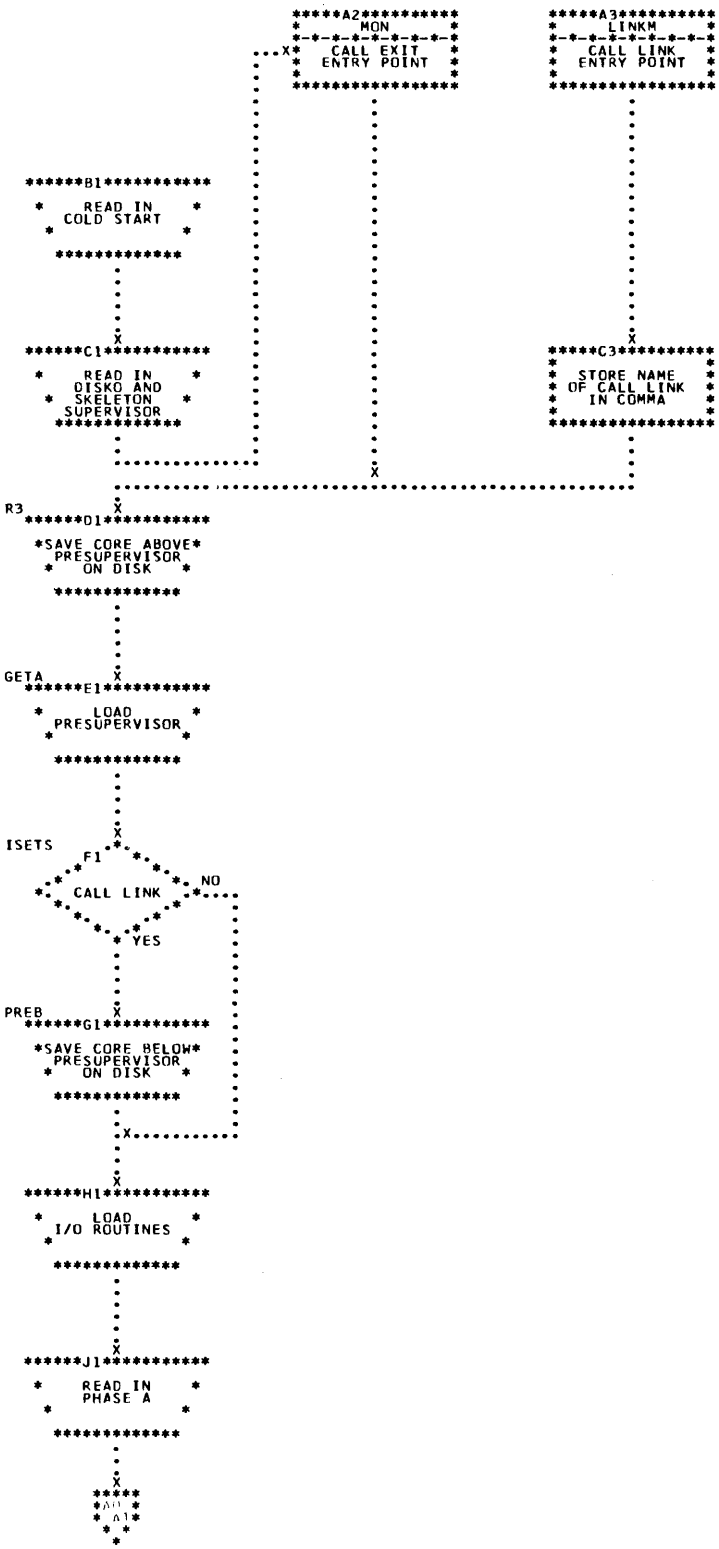


Chart AC. The Skelton Supervisor, Presupervisor, and Cold Start Routine

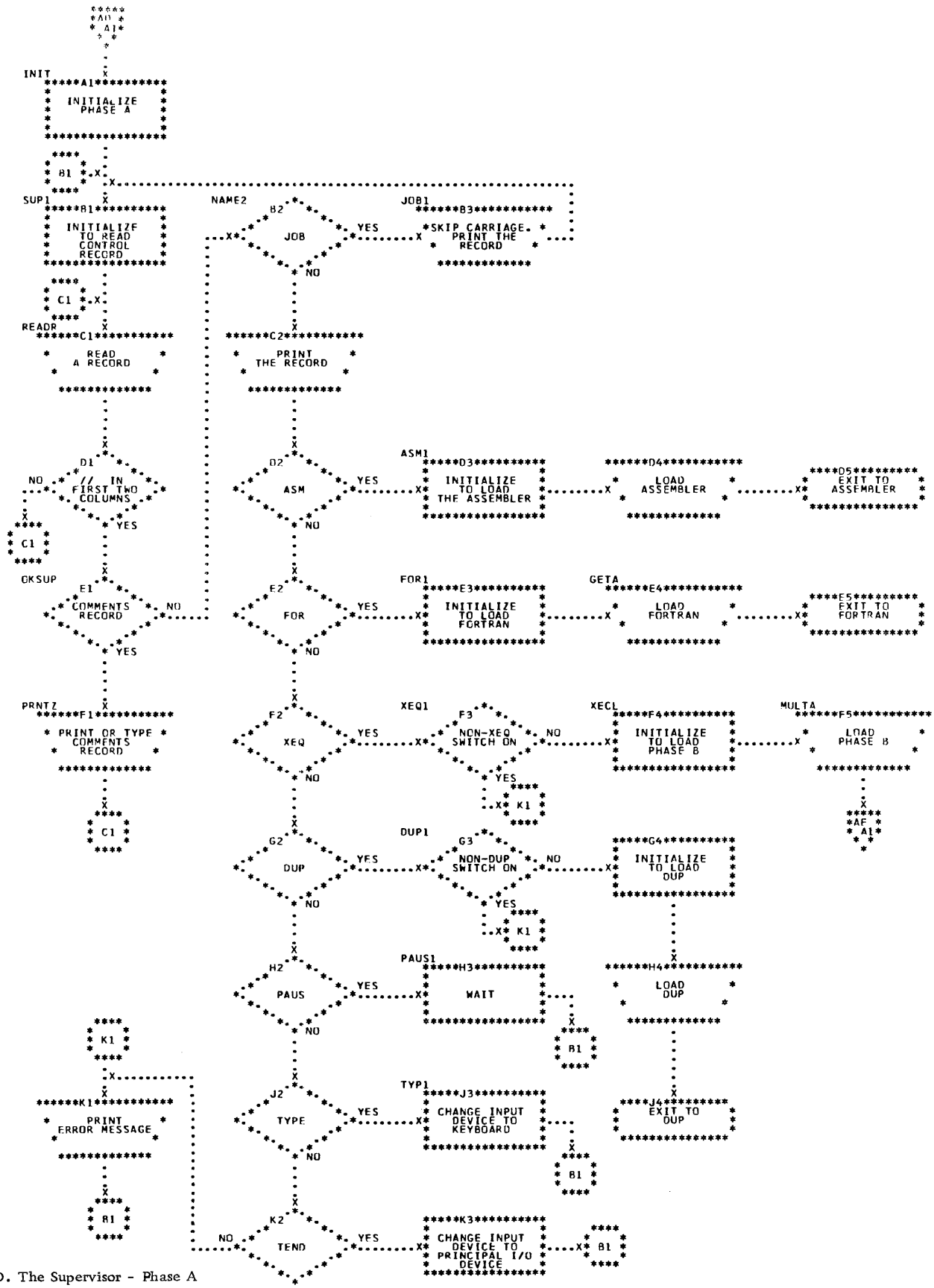


Chart AD. The Supervisor - Phase A

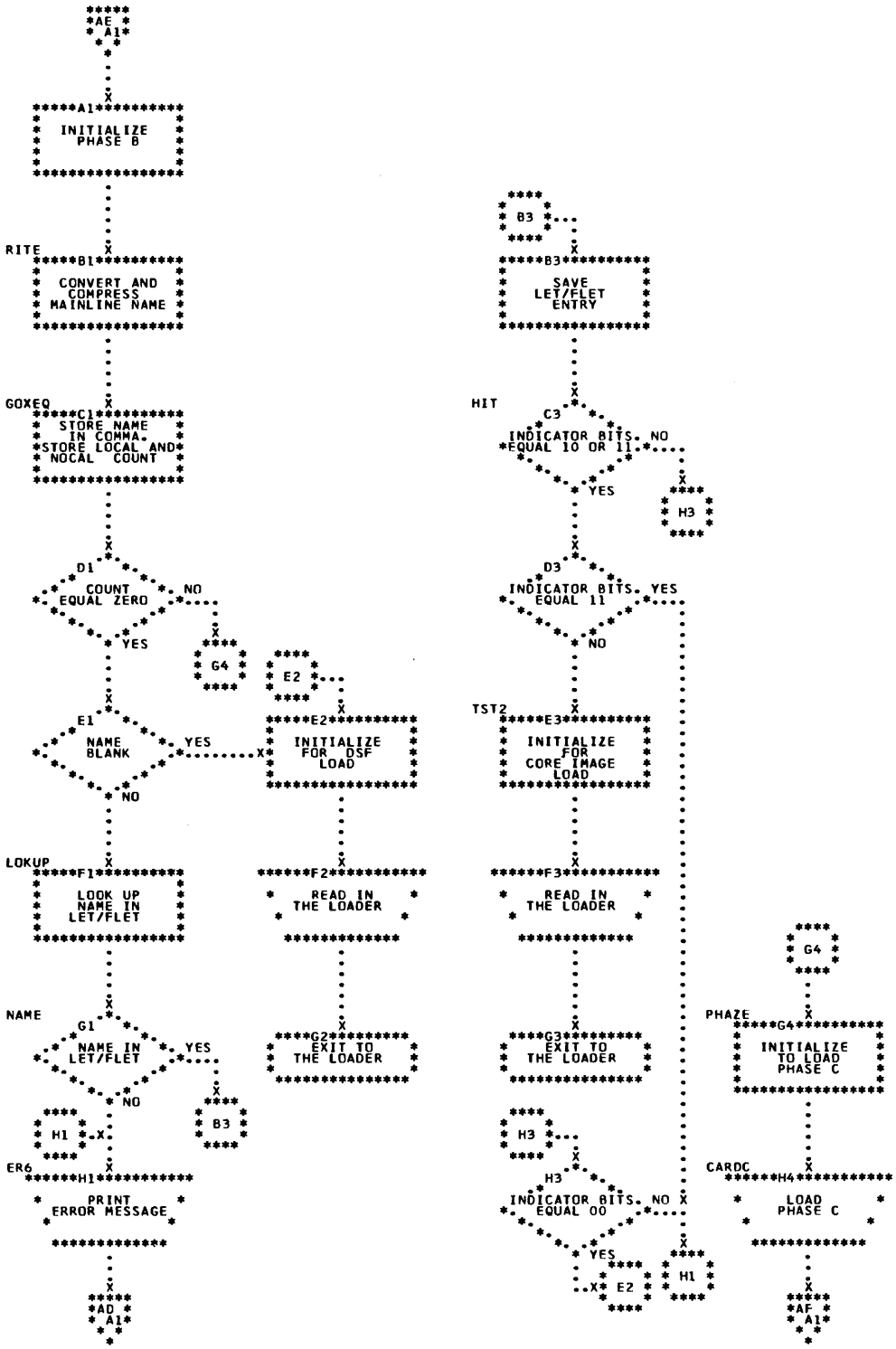


Chart AE. The Supervisor - Phase B

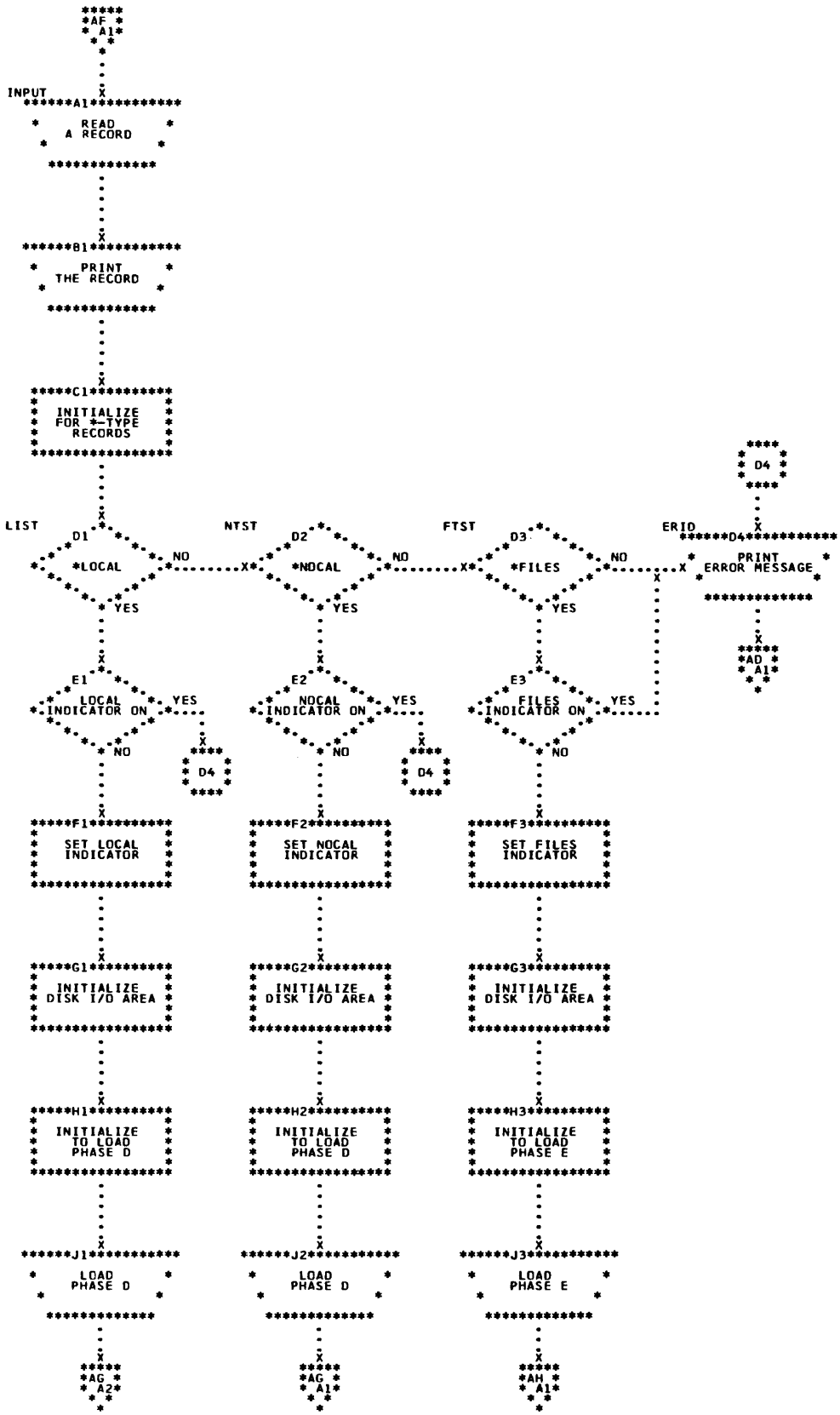


Chart AF. The Supervisor - Phase C

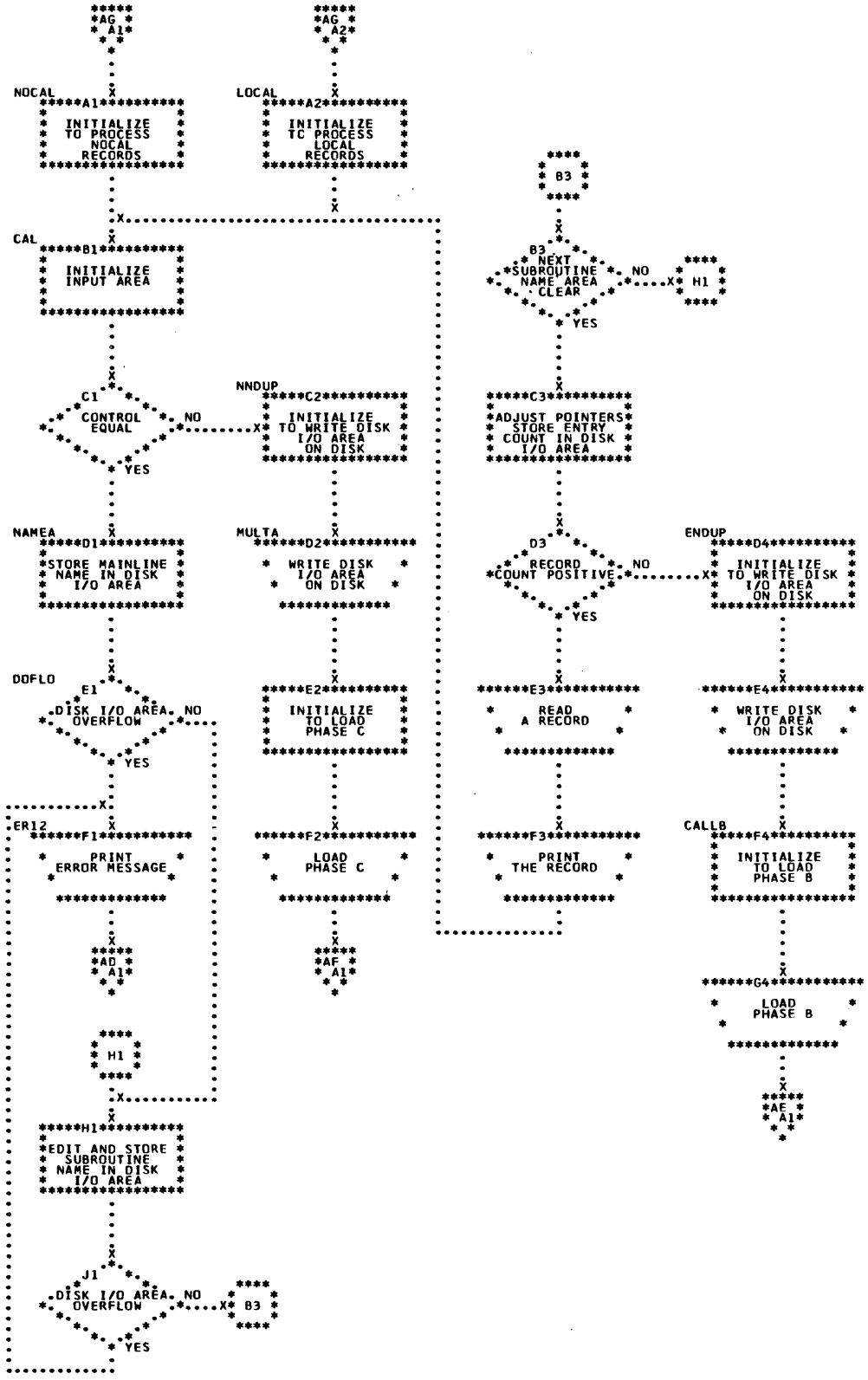


Chart AG. The Supervisor - Phase D

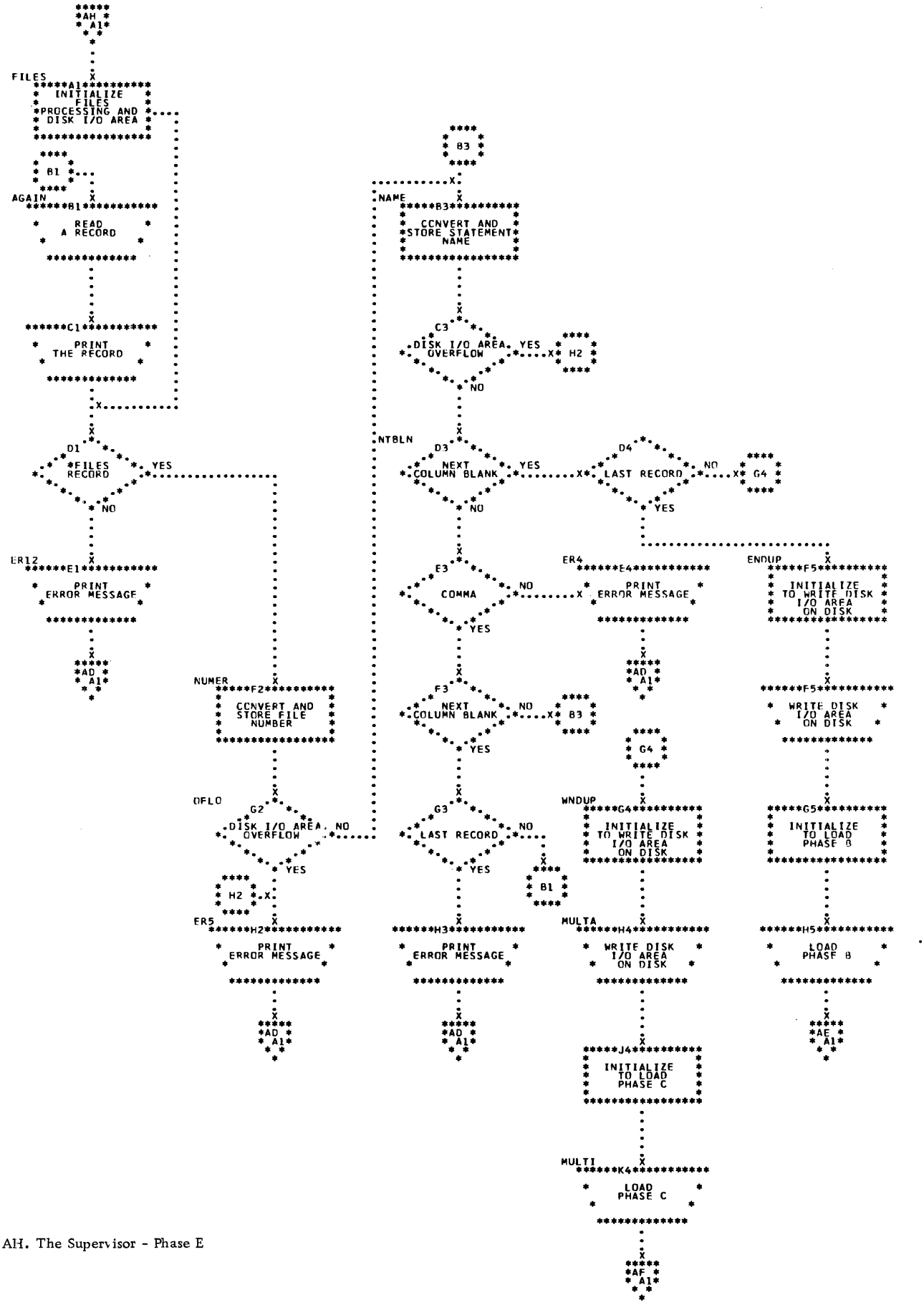


Chart AH. The Supervisor - Phase E

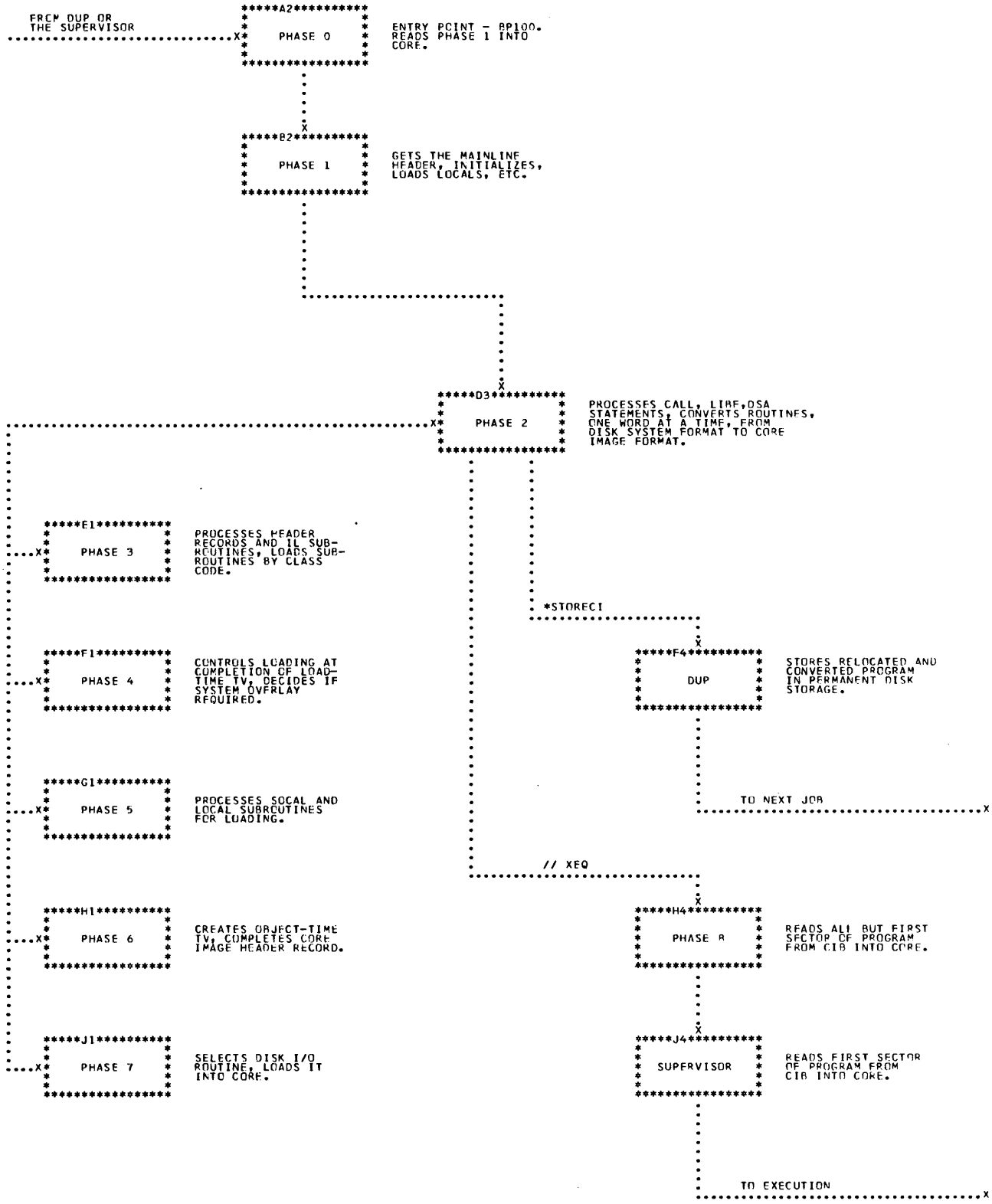


Chart AJ. The Loader - Disk System Format Load

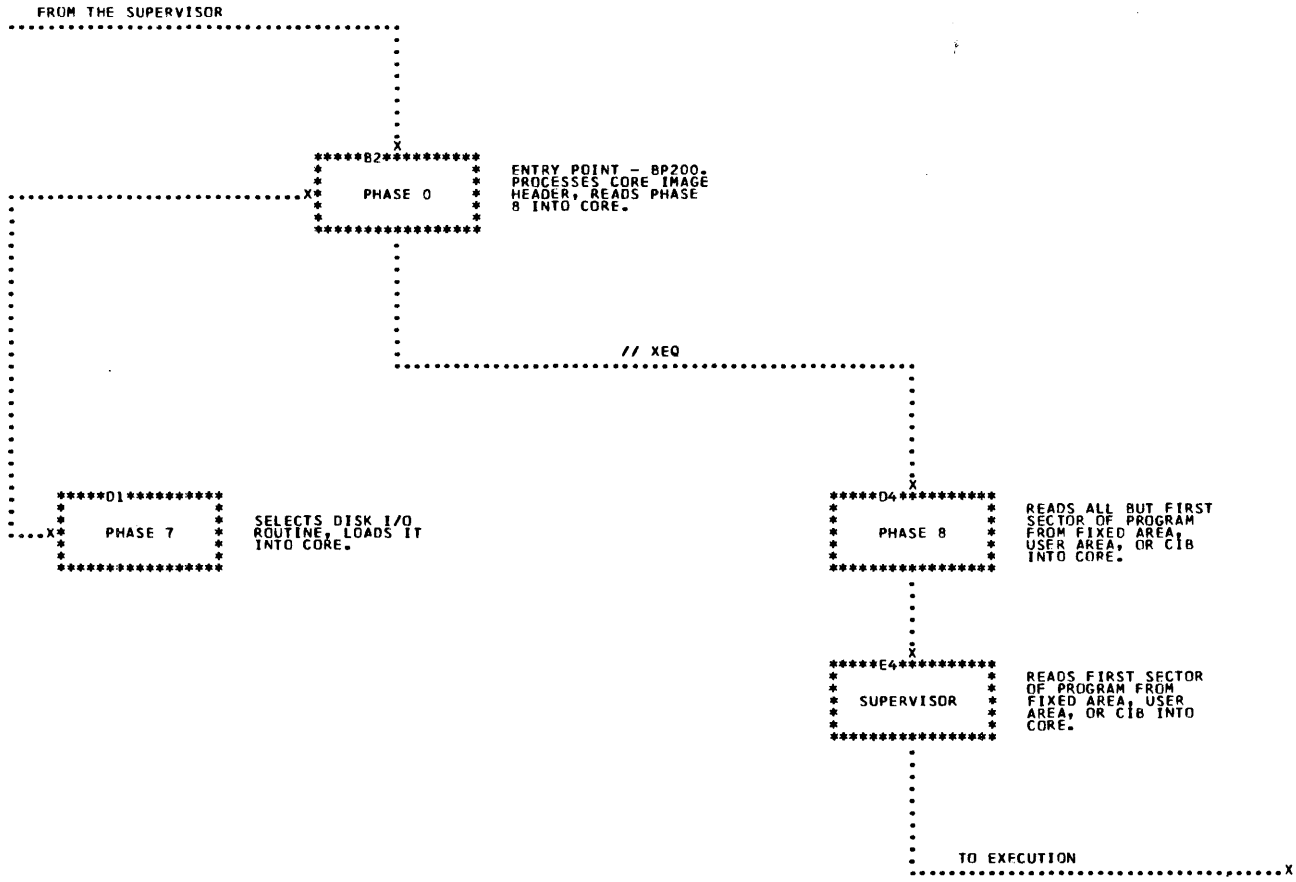


Chart AK. The Loader - Core Image Format Load

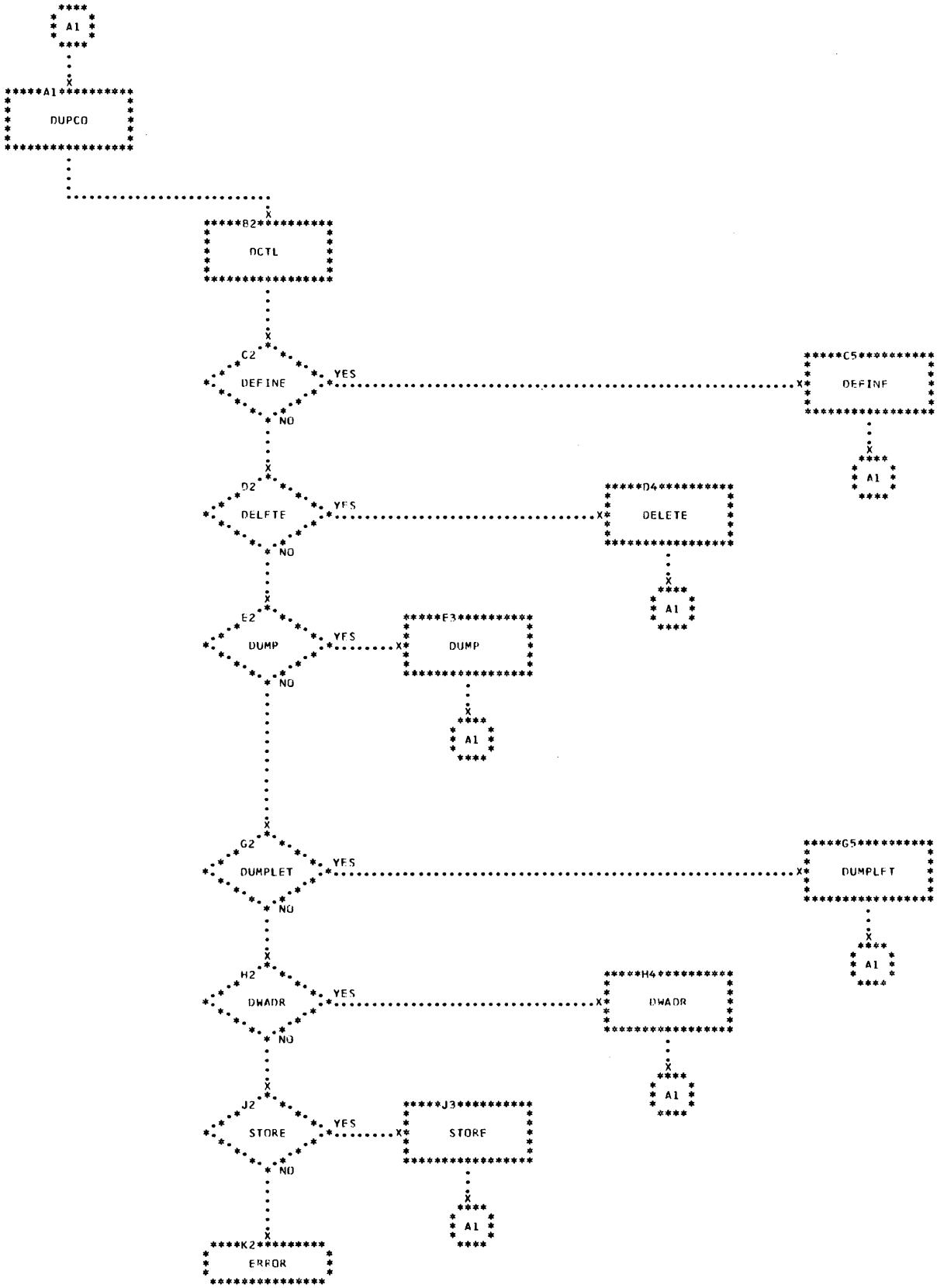


Chart BA. DUP Functions

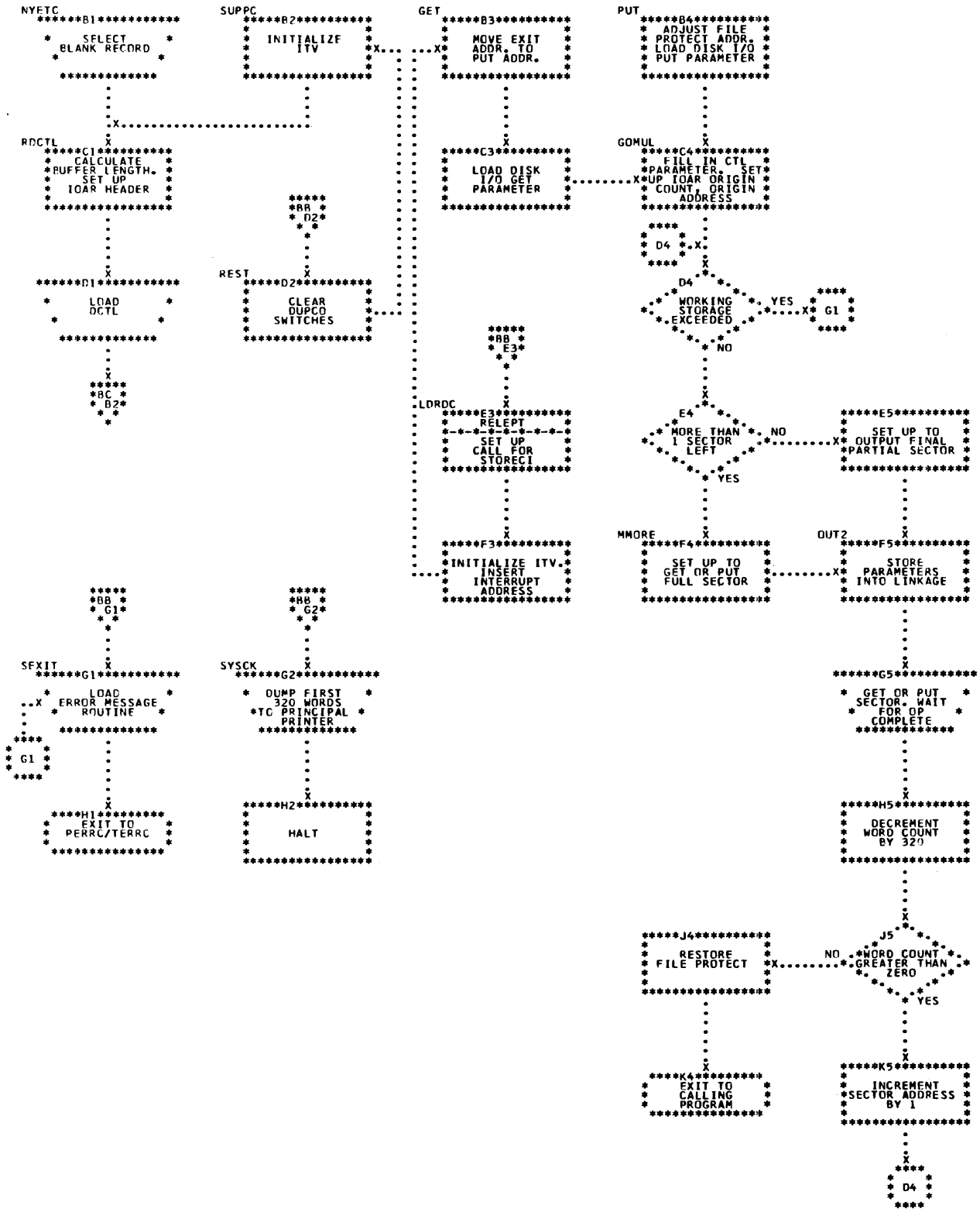


Chart BB. DUP - DUPCO

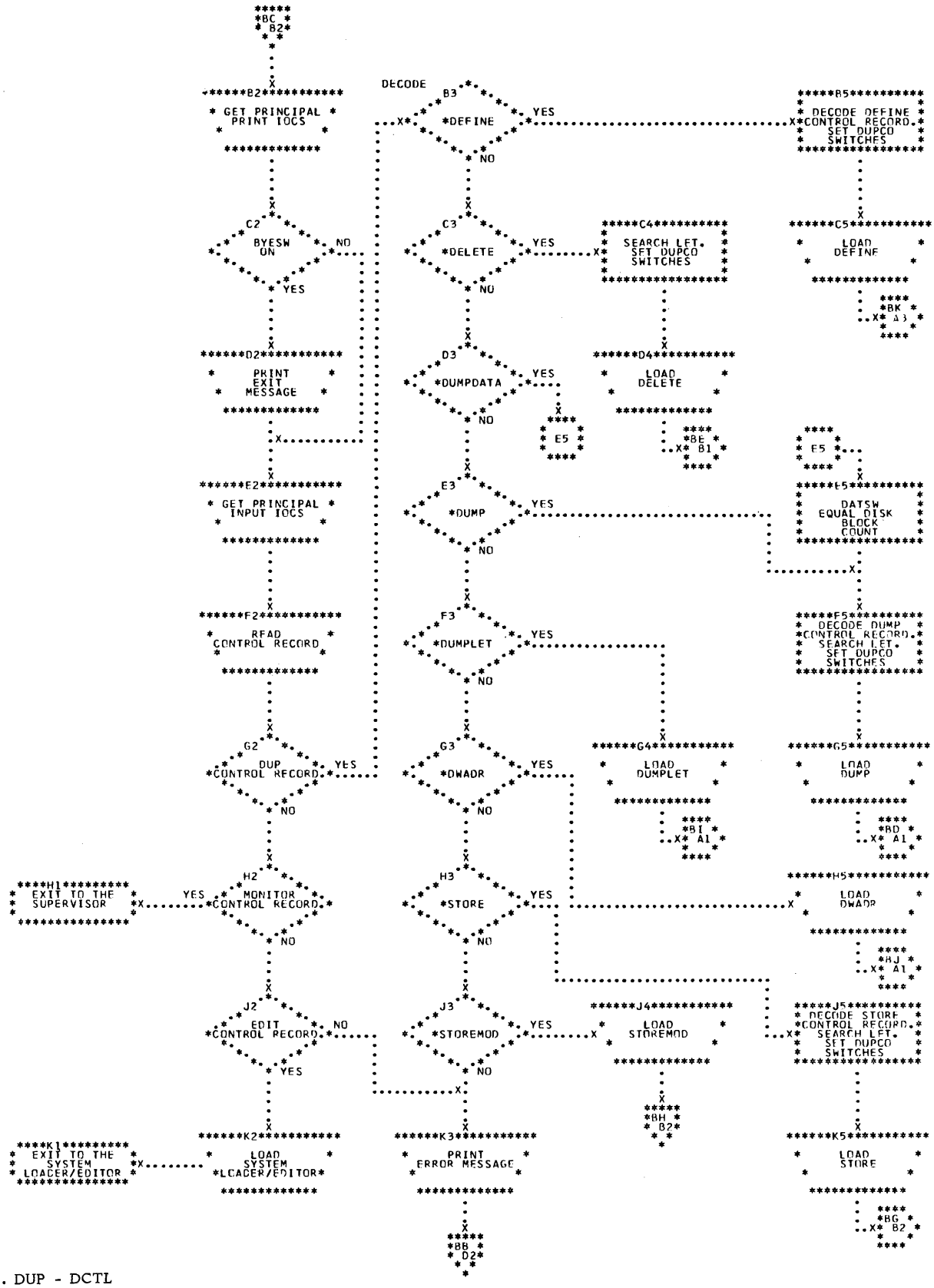


Chart BC. DUP - DCTL

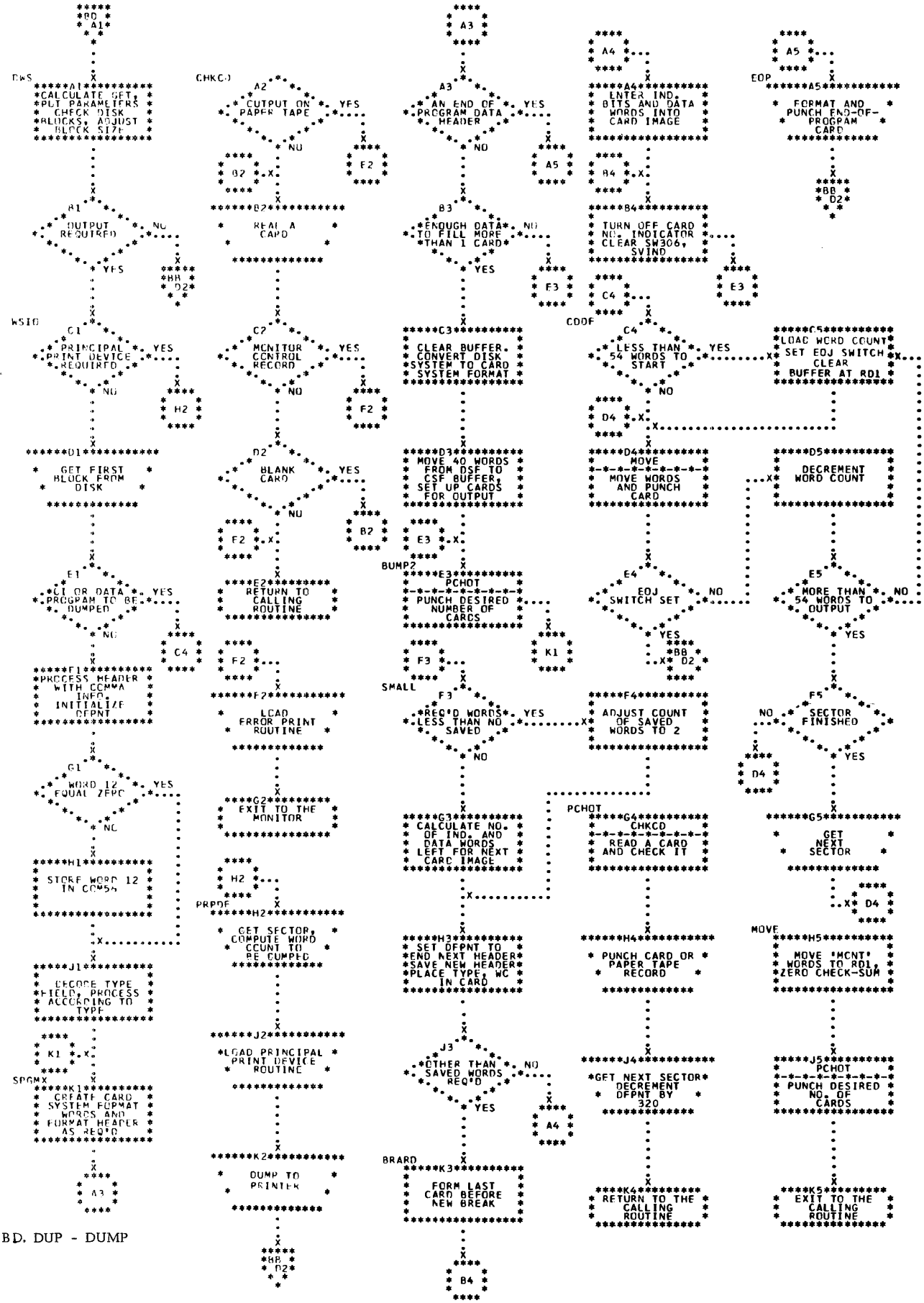


Chart BD. DUP - DUMP

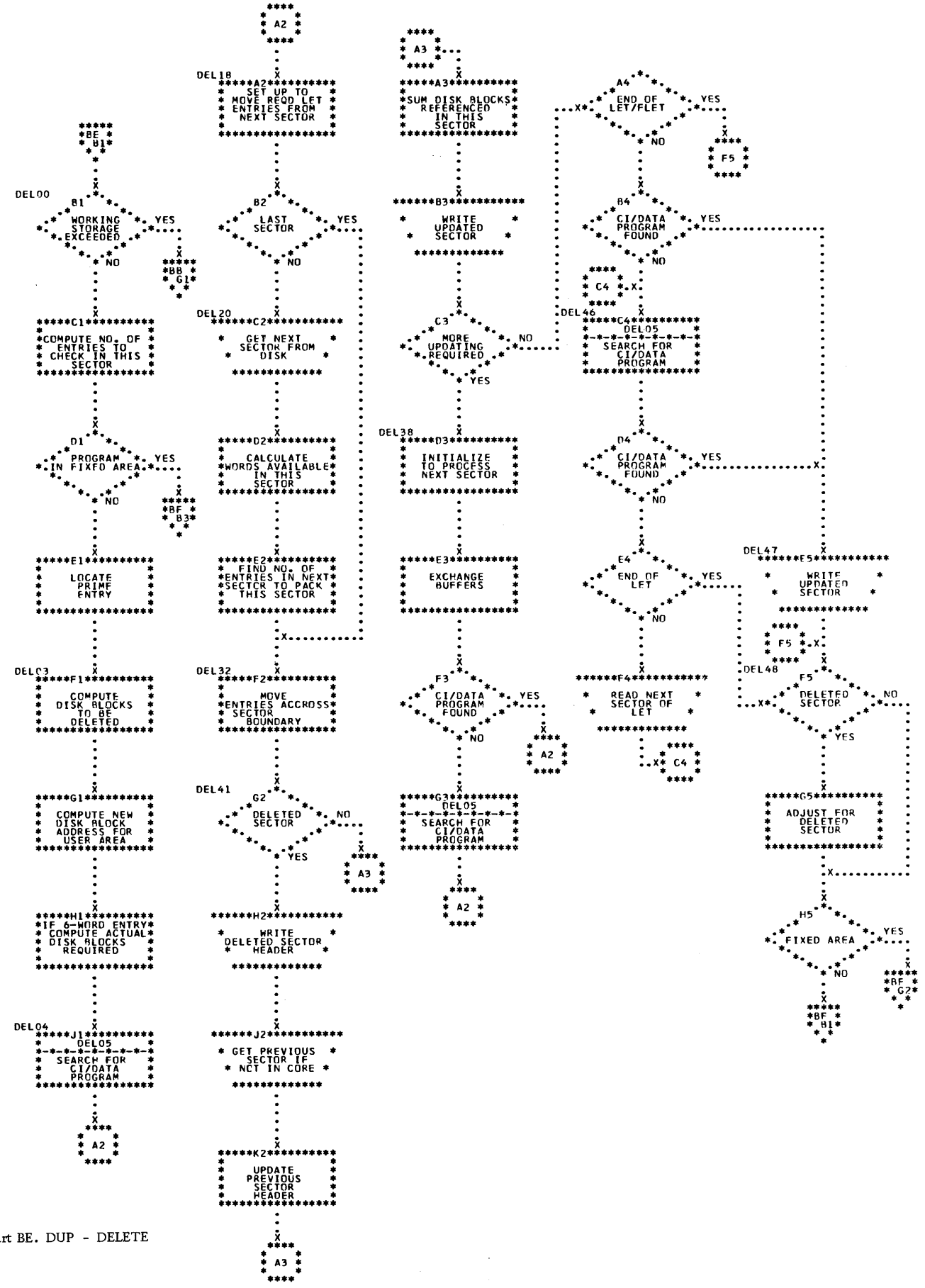


Chart BE. DUP - DELETE

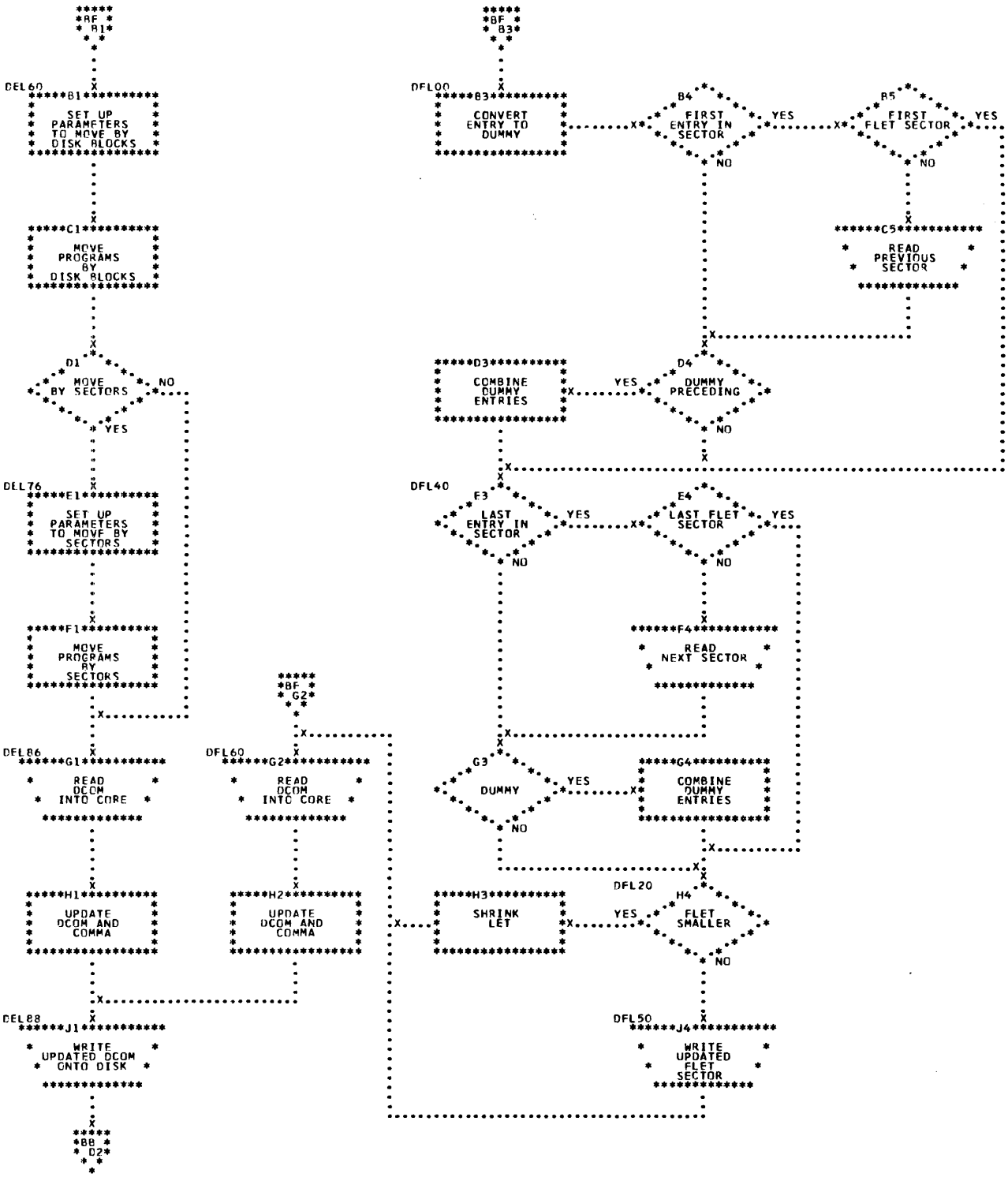


Chart BF. DUP - DELETE

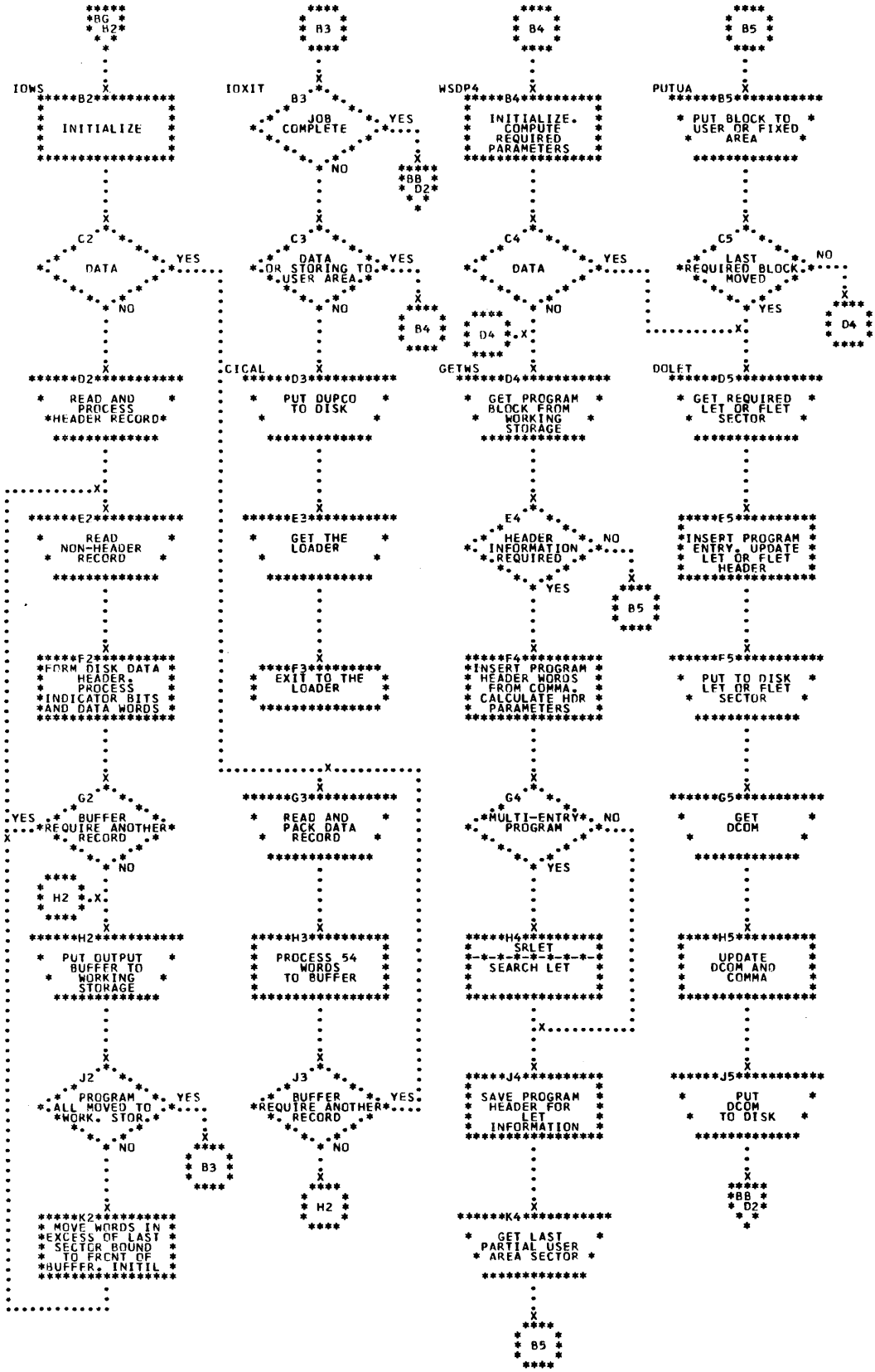


Chart BG. DUP - STORE

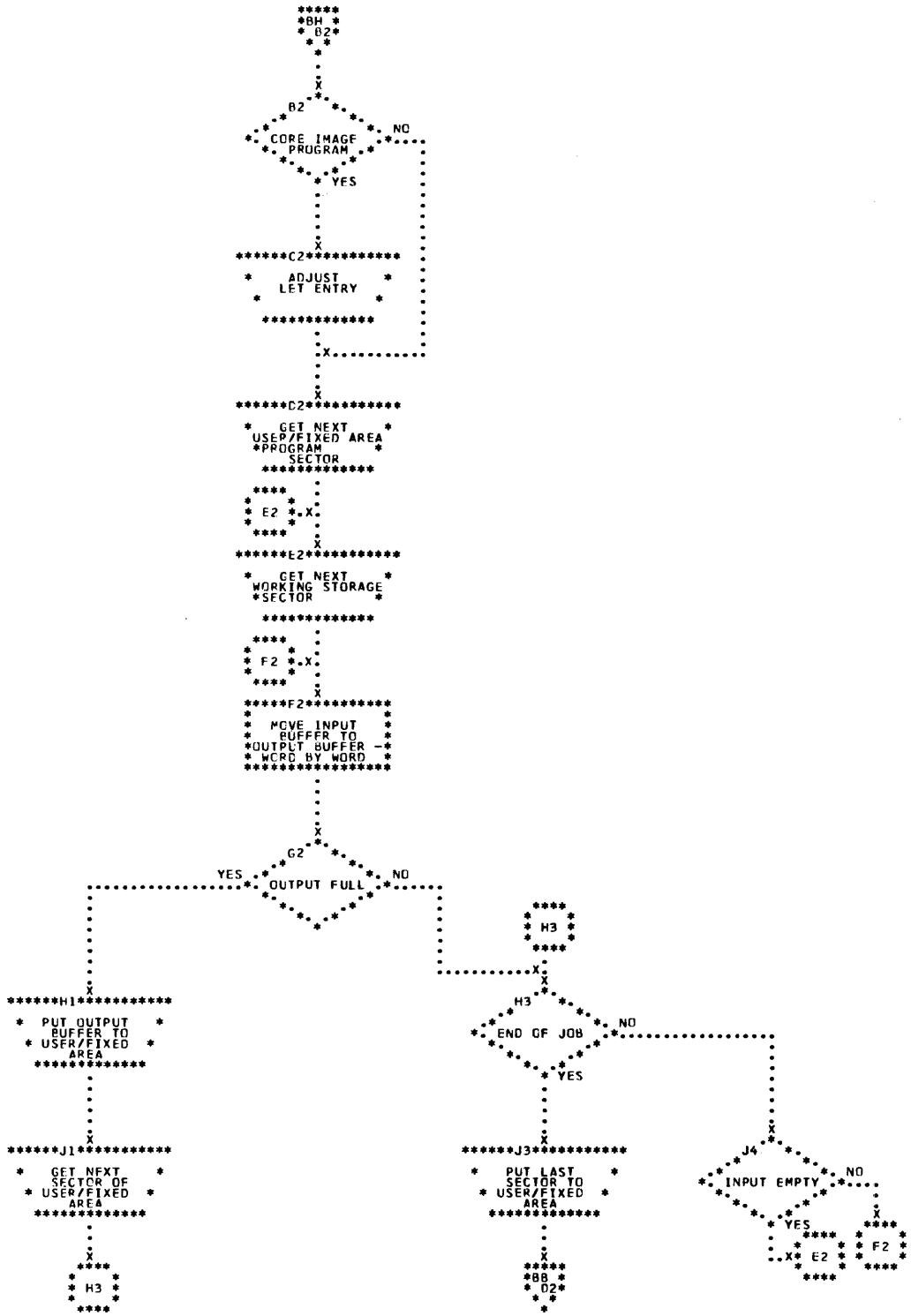


Chart BH. DUP - STOREMOD

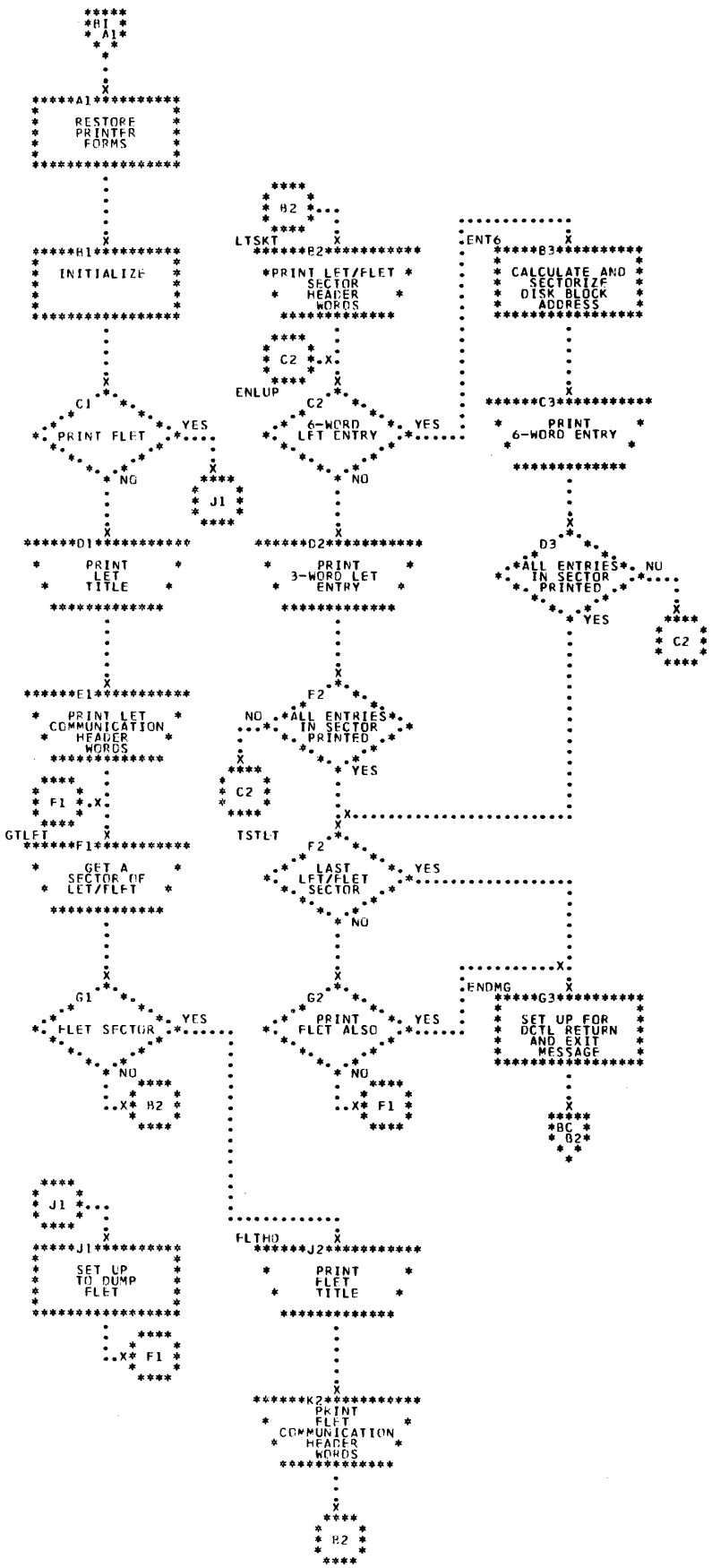


Chart BI. DUP - DUMPLET

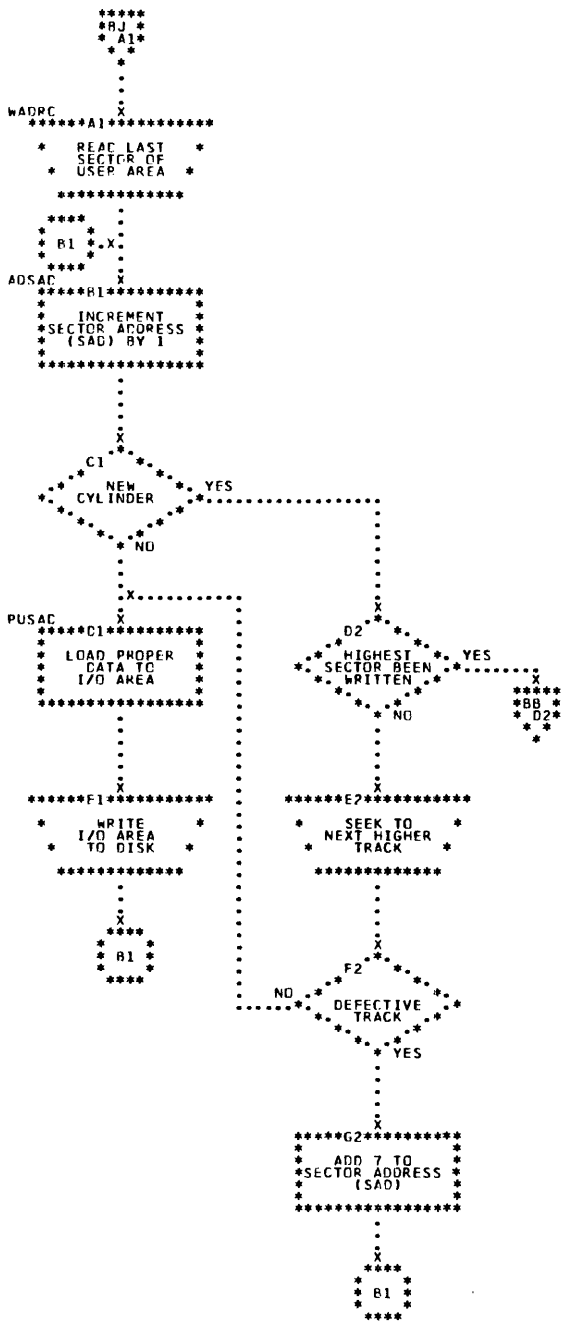


Chart BJ. DUP - DWADR

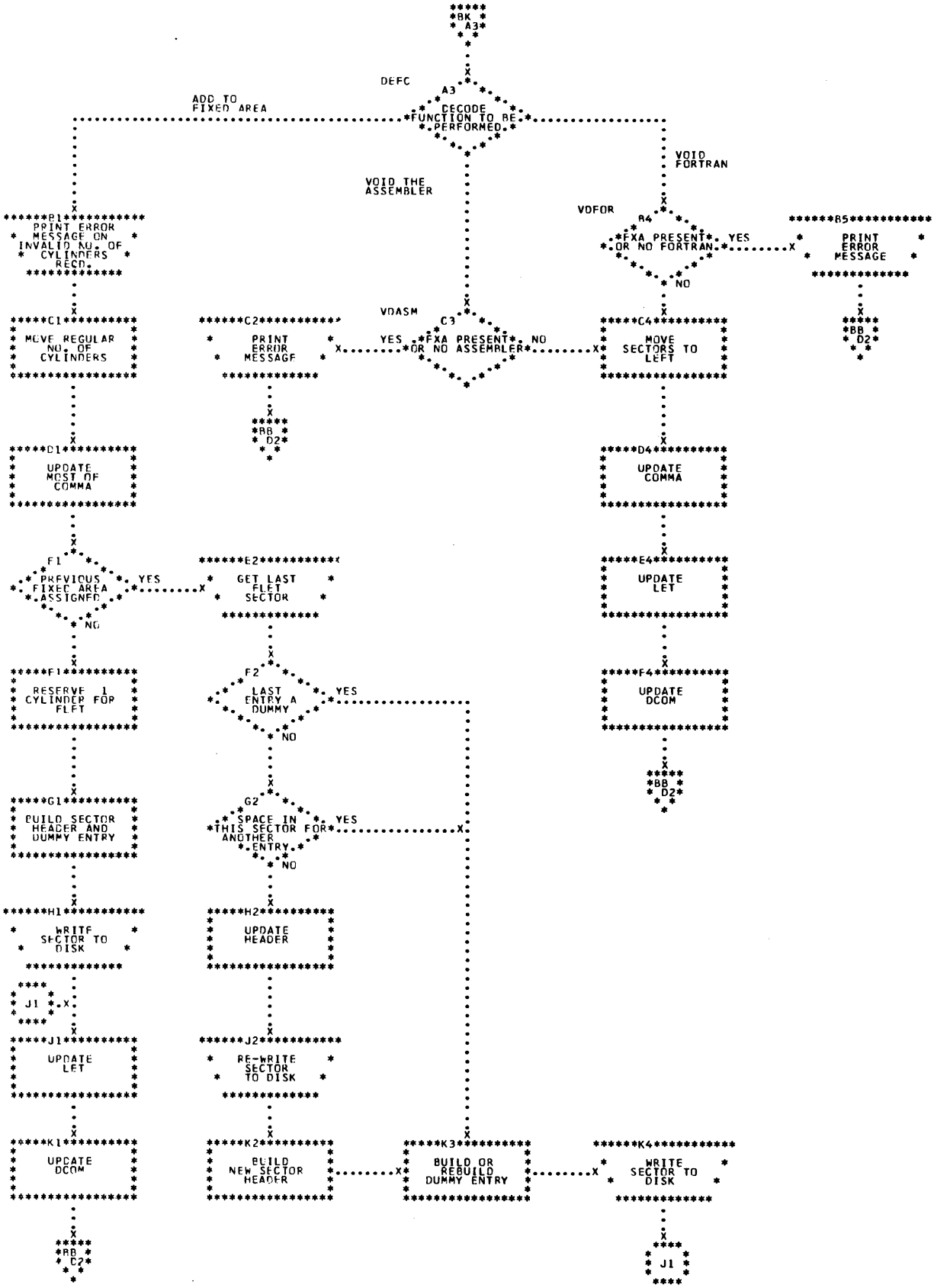


Chart BK. DUP - DEFINE

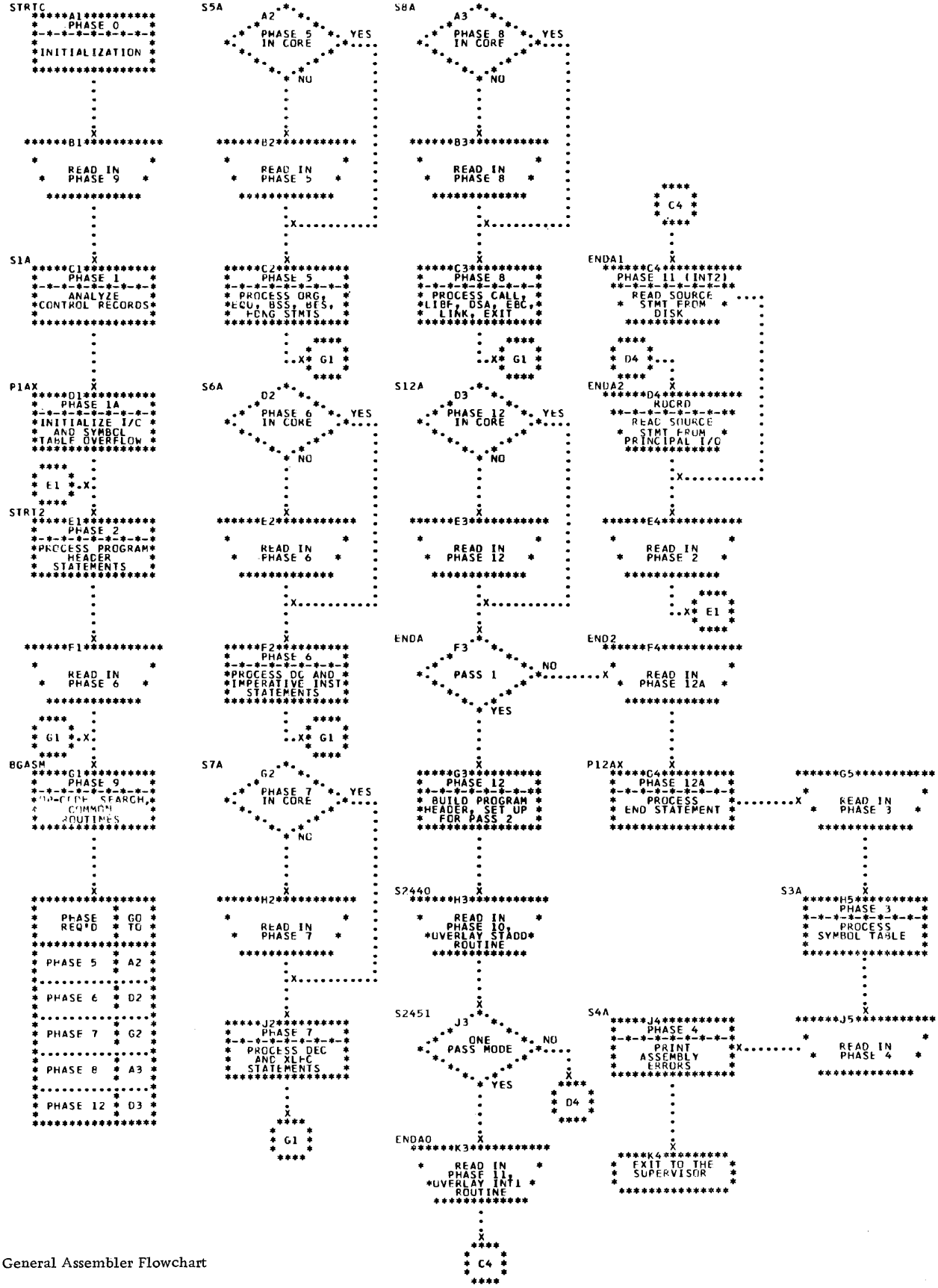


Chart BL. General Assembler Flowchart

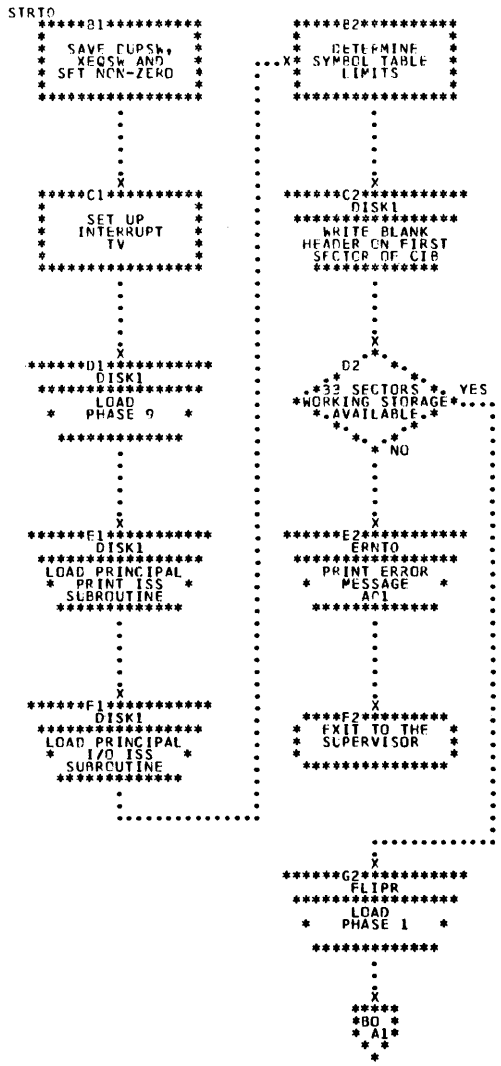


Chart BM. The Assembler - Phase 0

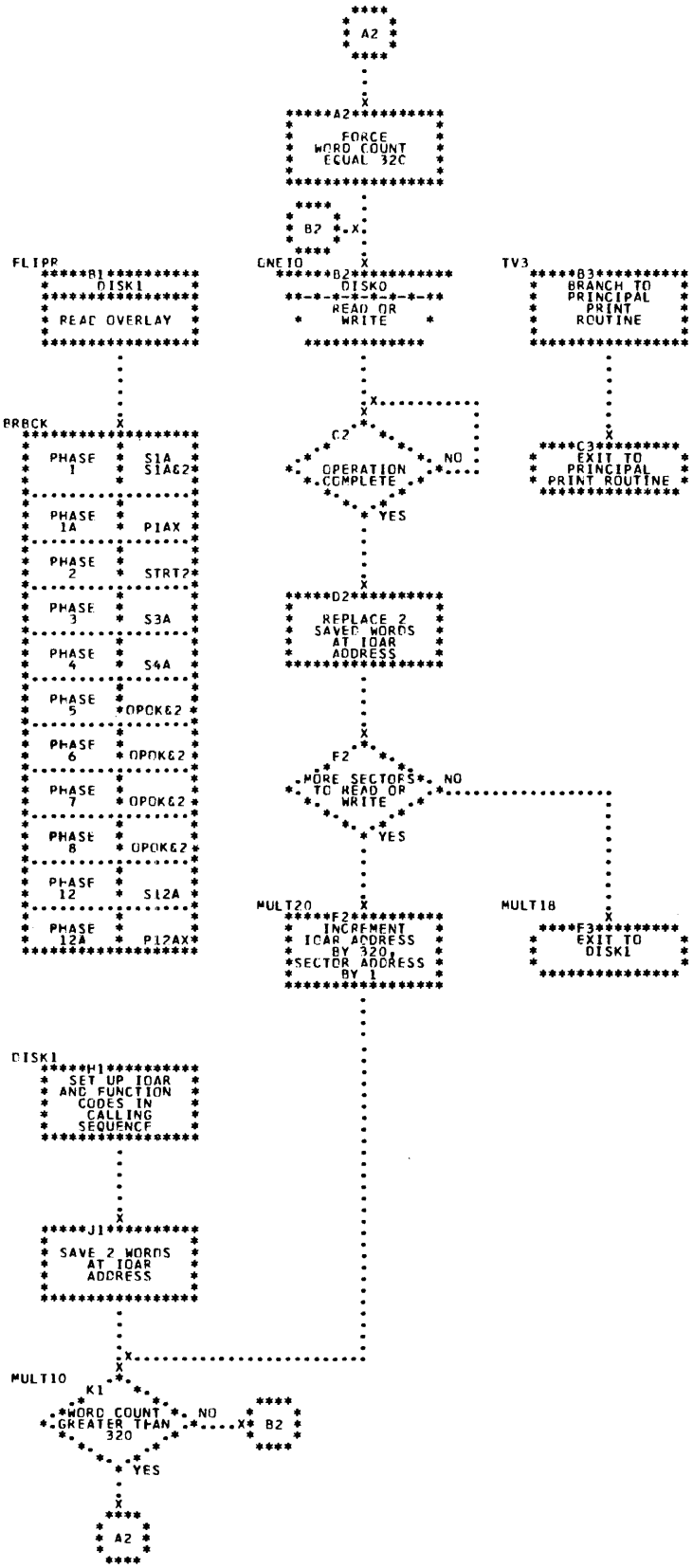


Chart BN. The Assembler - Phase 0

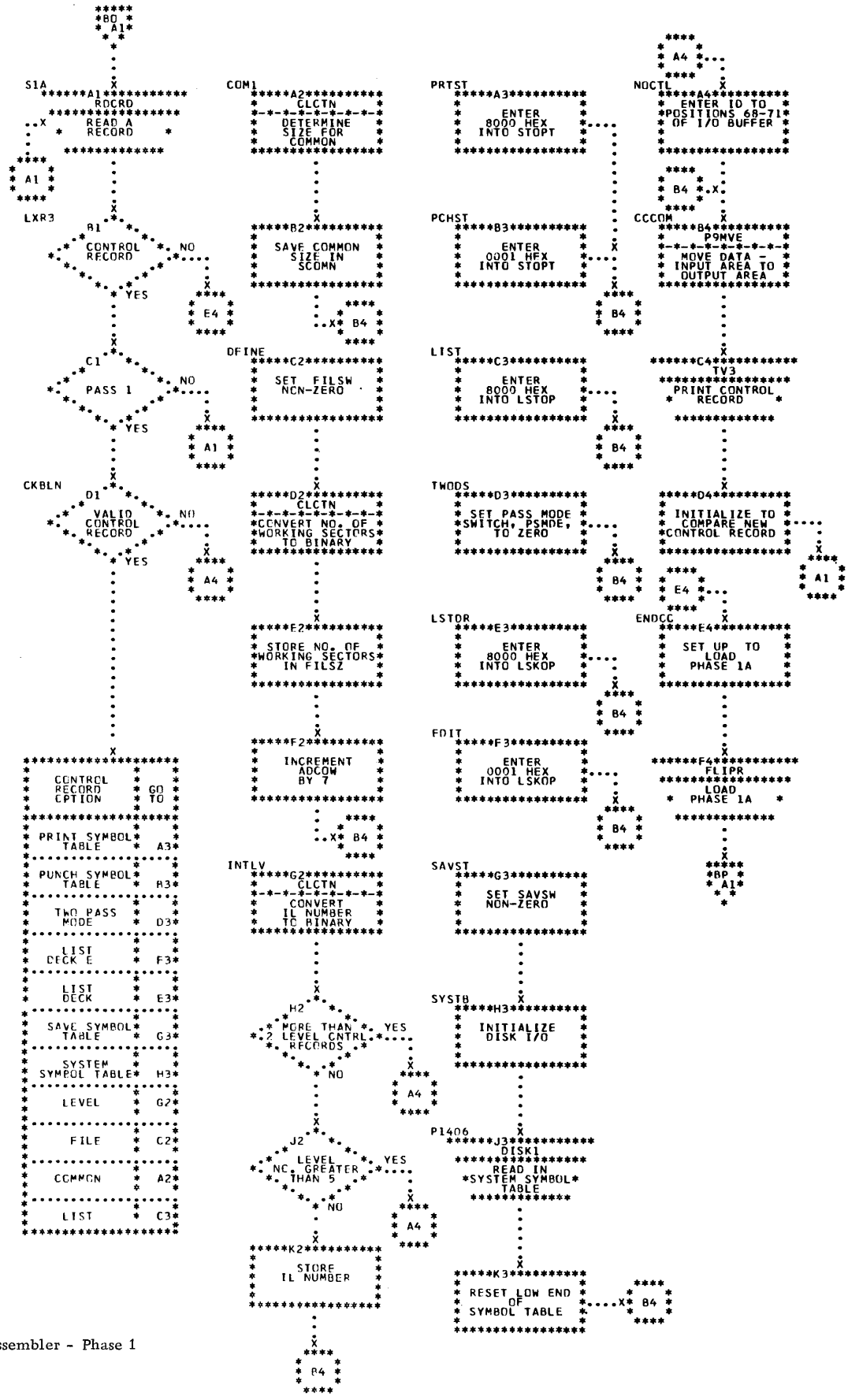


Chart BO. The Assembler - Phase 1

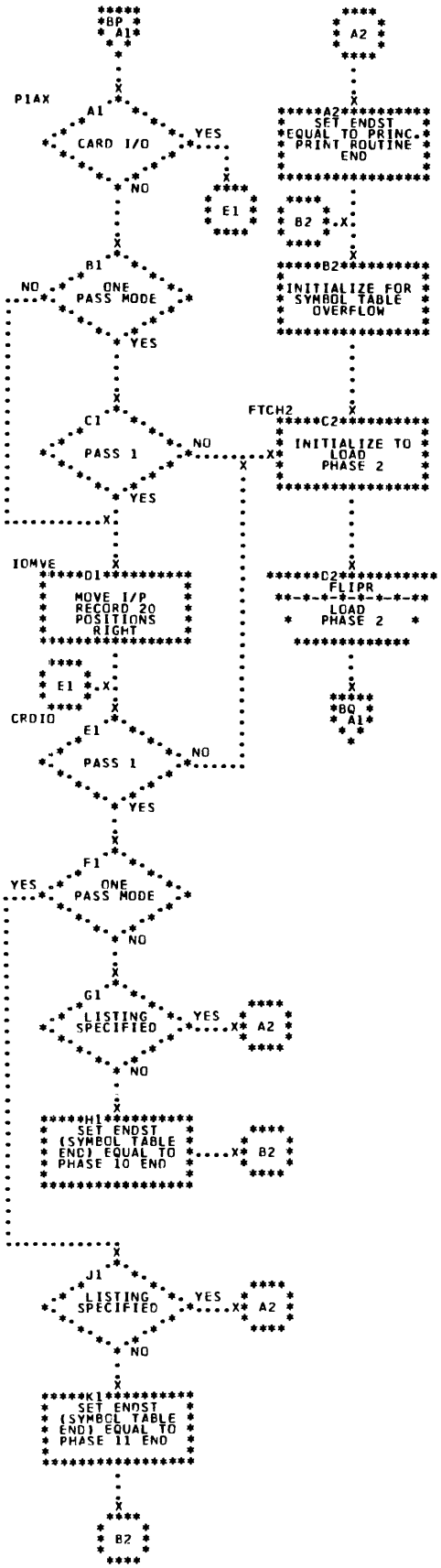


Chart BP. The Assembler - Phase 1A

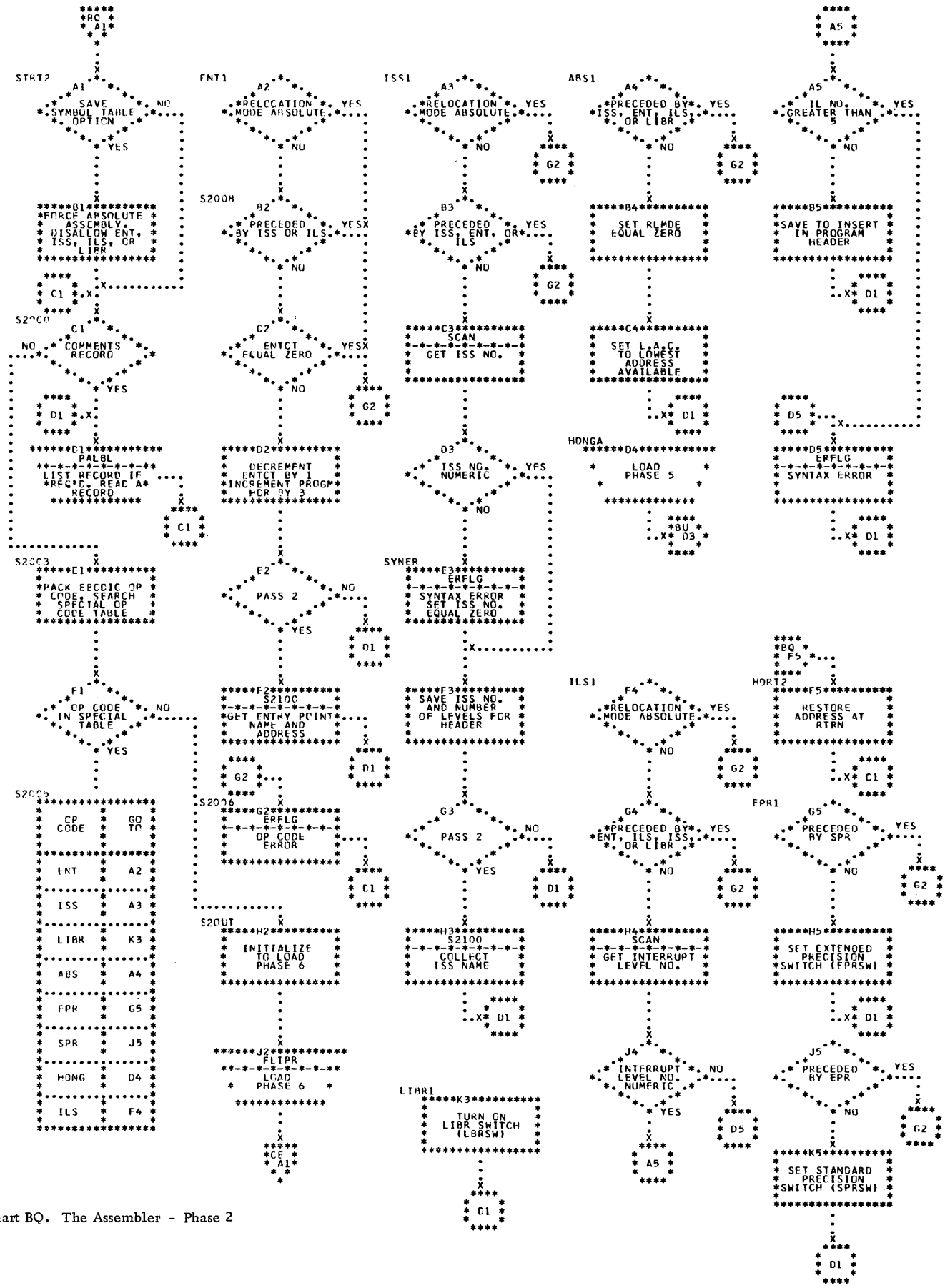


Chart BQ. The Assembler - Phase 2

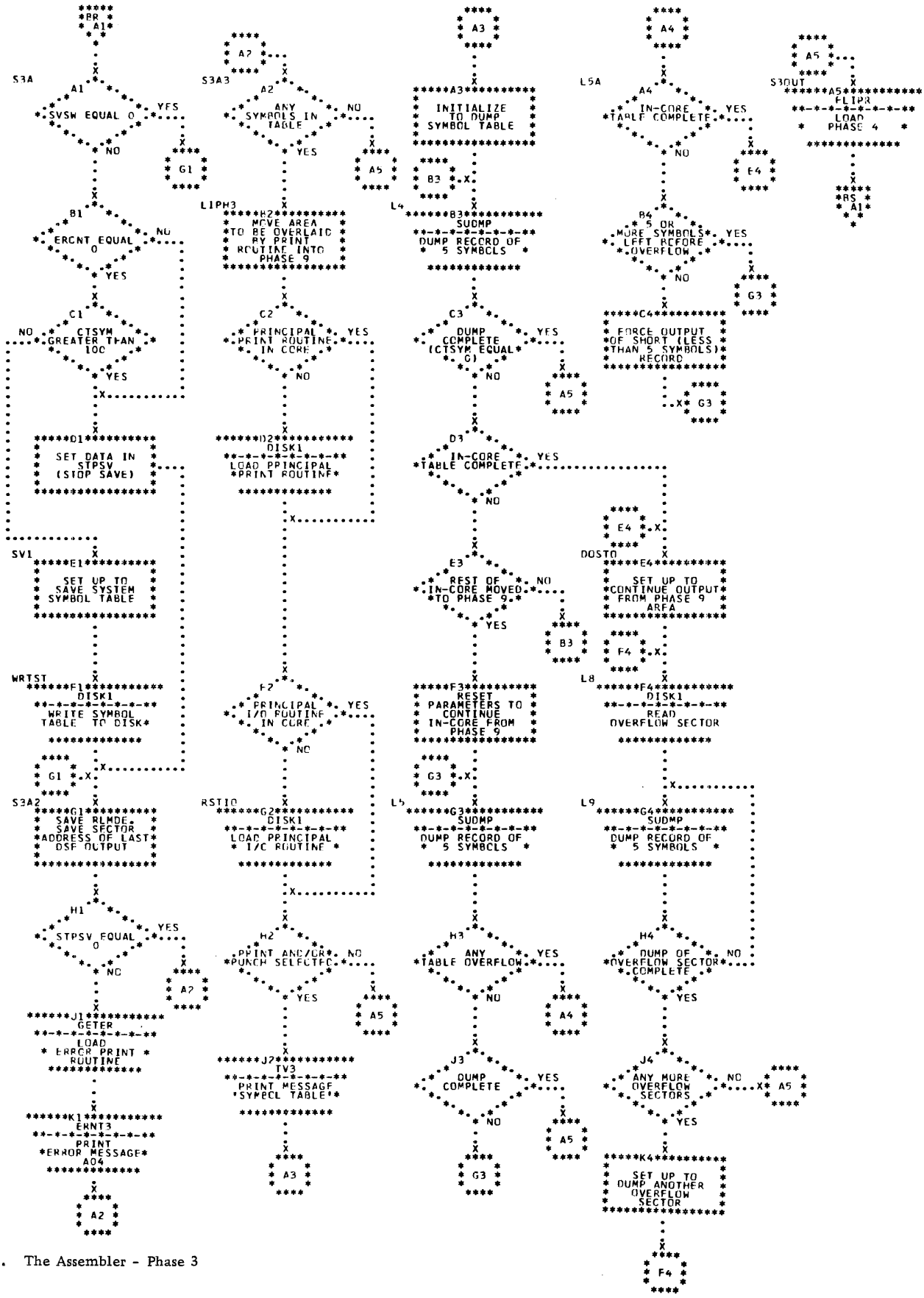


Chart BR. The Assembler - Phase 3

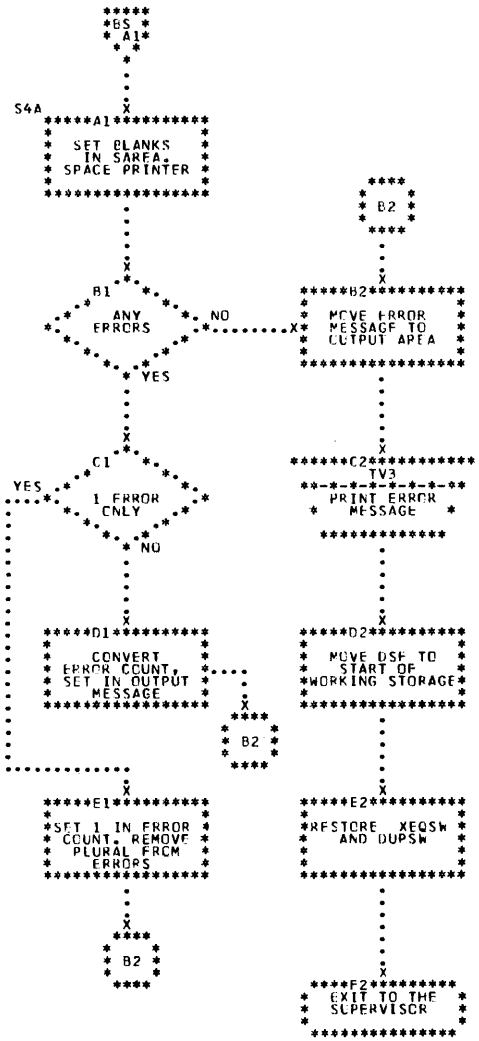


Chart BS. The Assembler - Phase 4

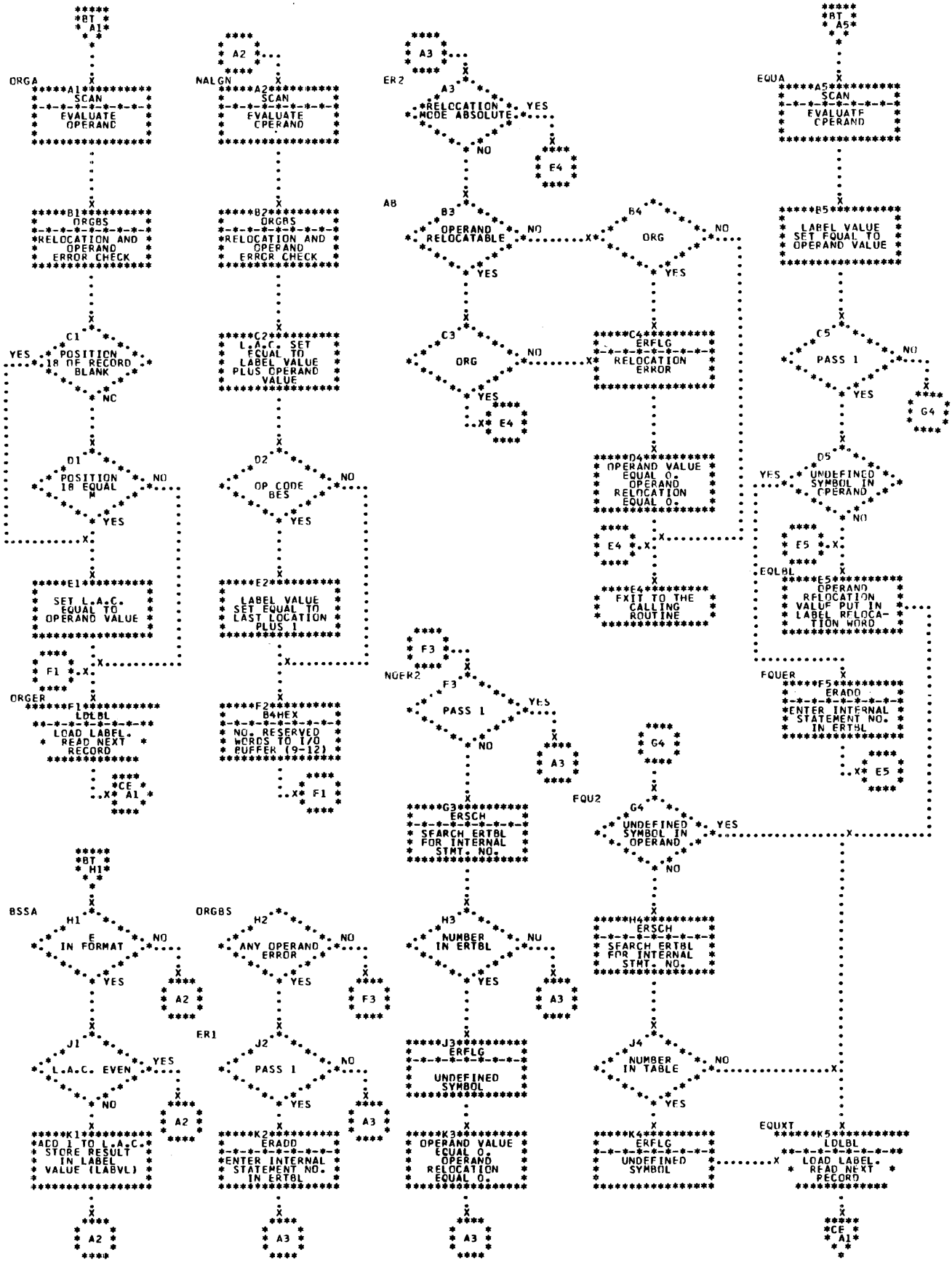


Chart BT. The Assembler - Phase 5

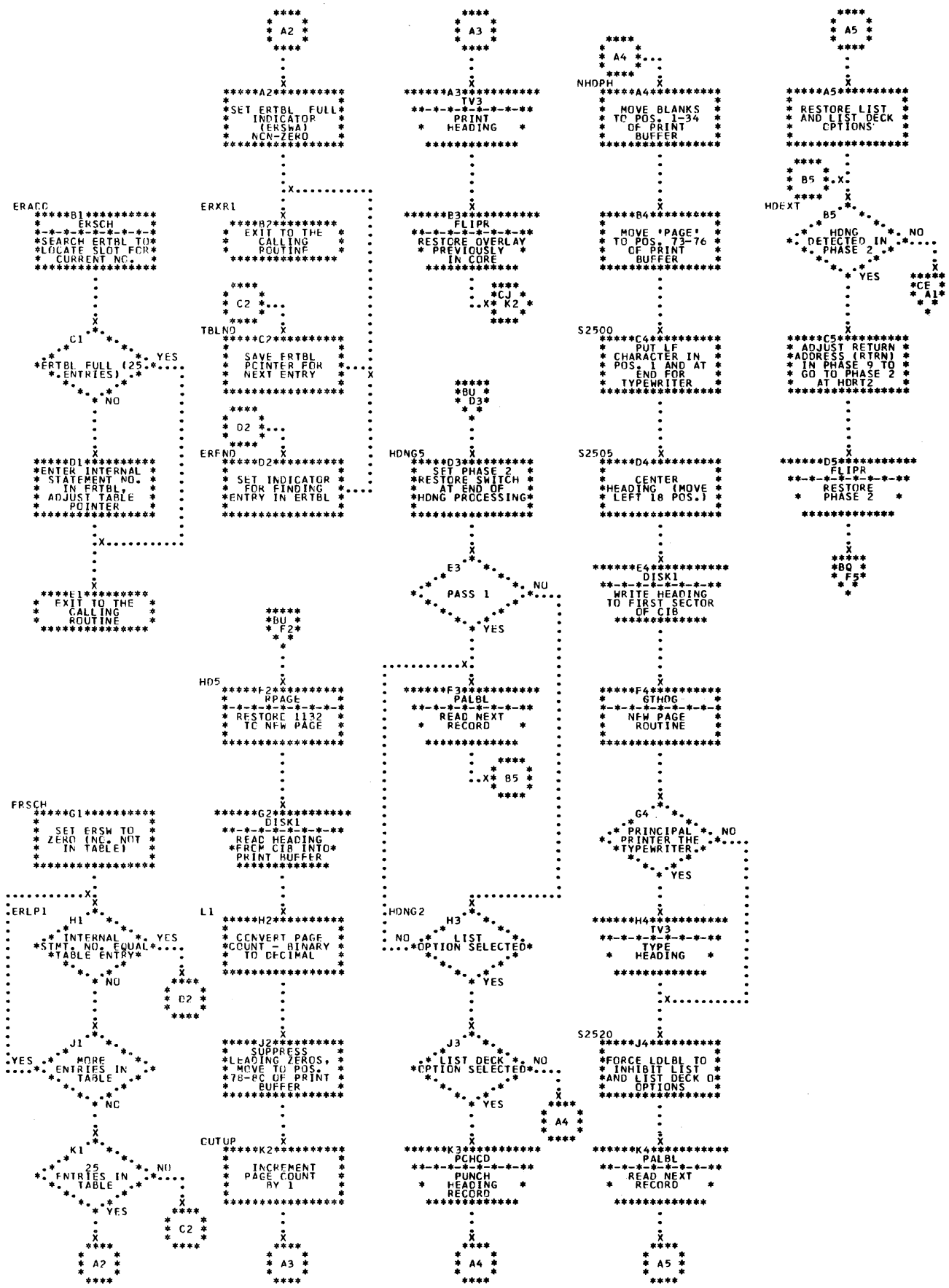


Chart BU. The Assembler - Phase 5

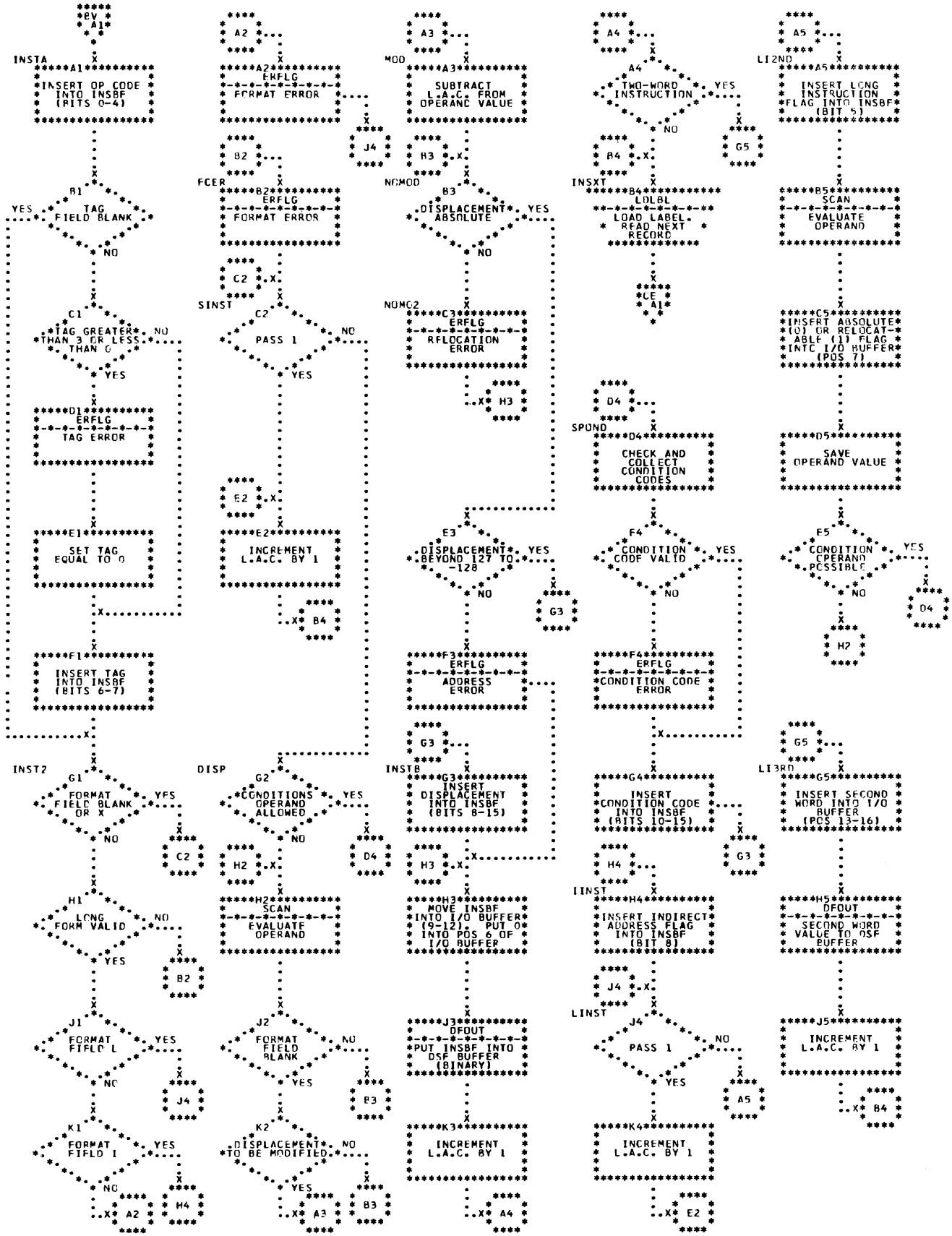


Chart BV. The Assembler - Phase 6

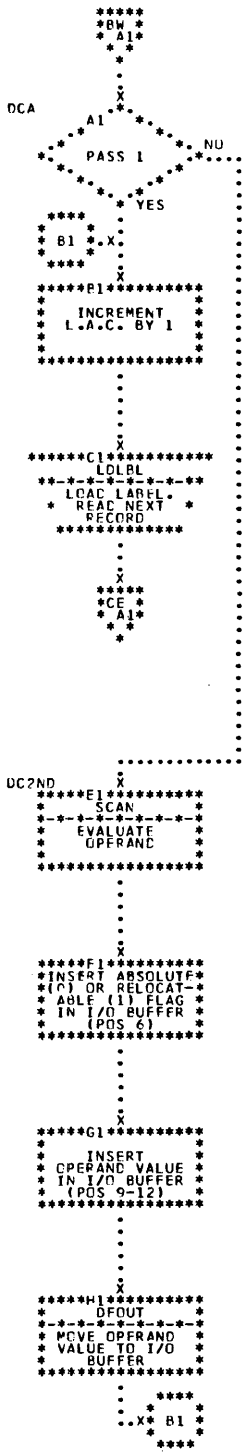


Chart BW. The Assembler - Phase 6

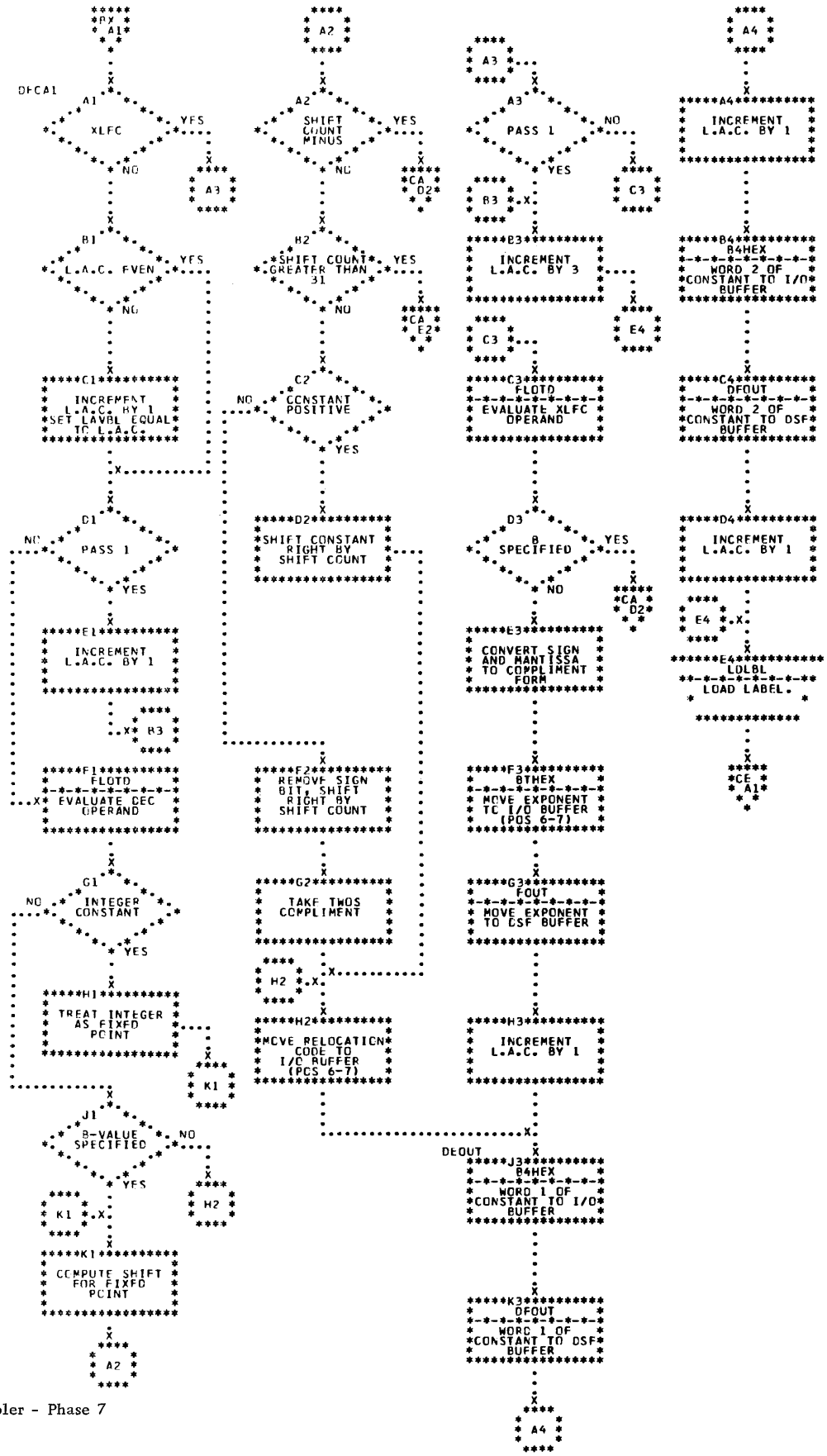


Chart BX. The Assembler - Phase 7

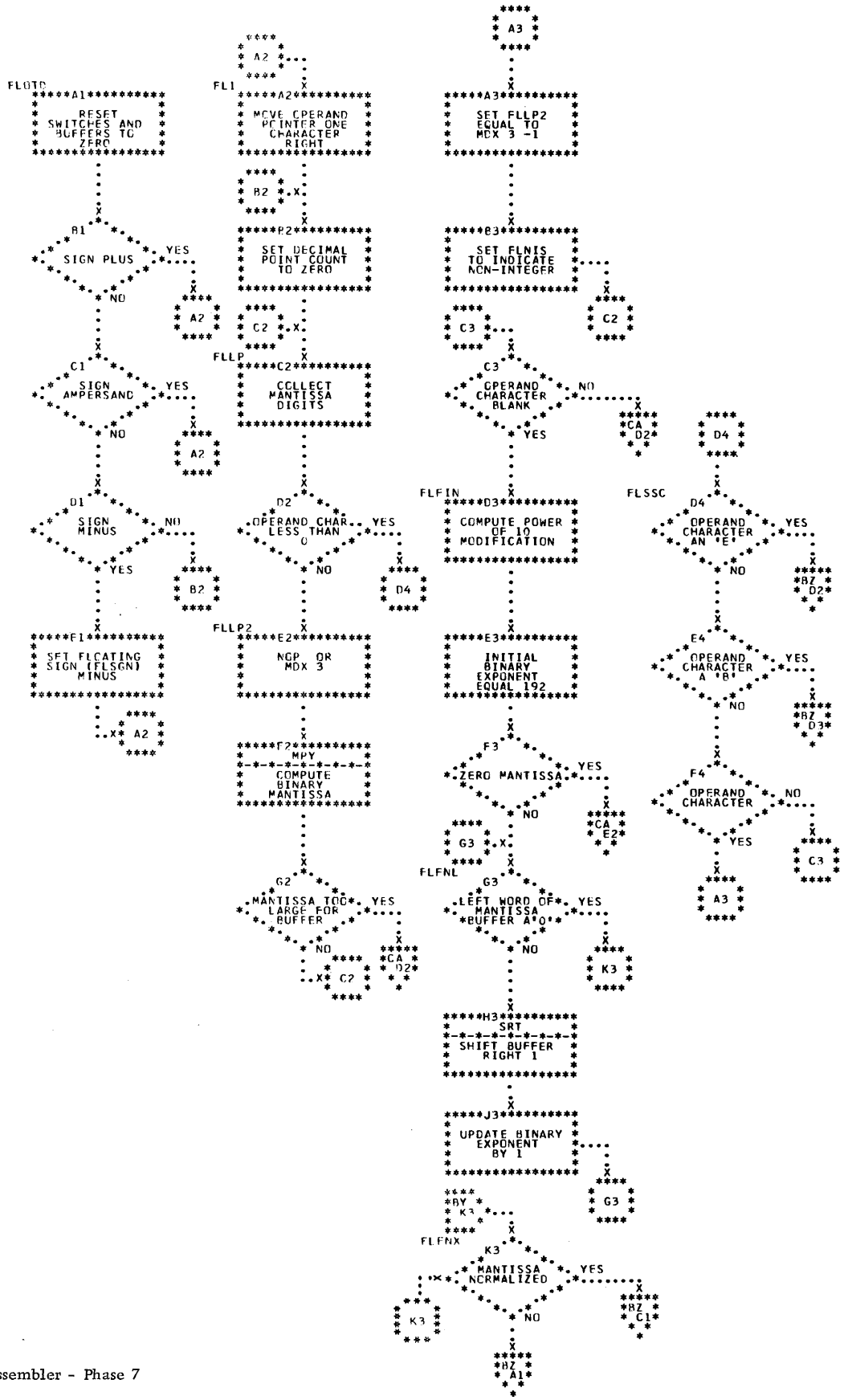


Chart BY. The Assembler - Phase 7

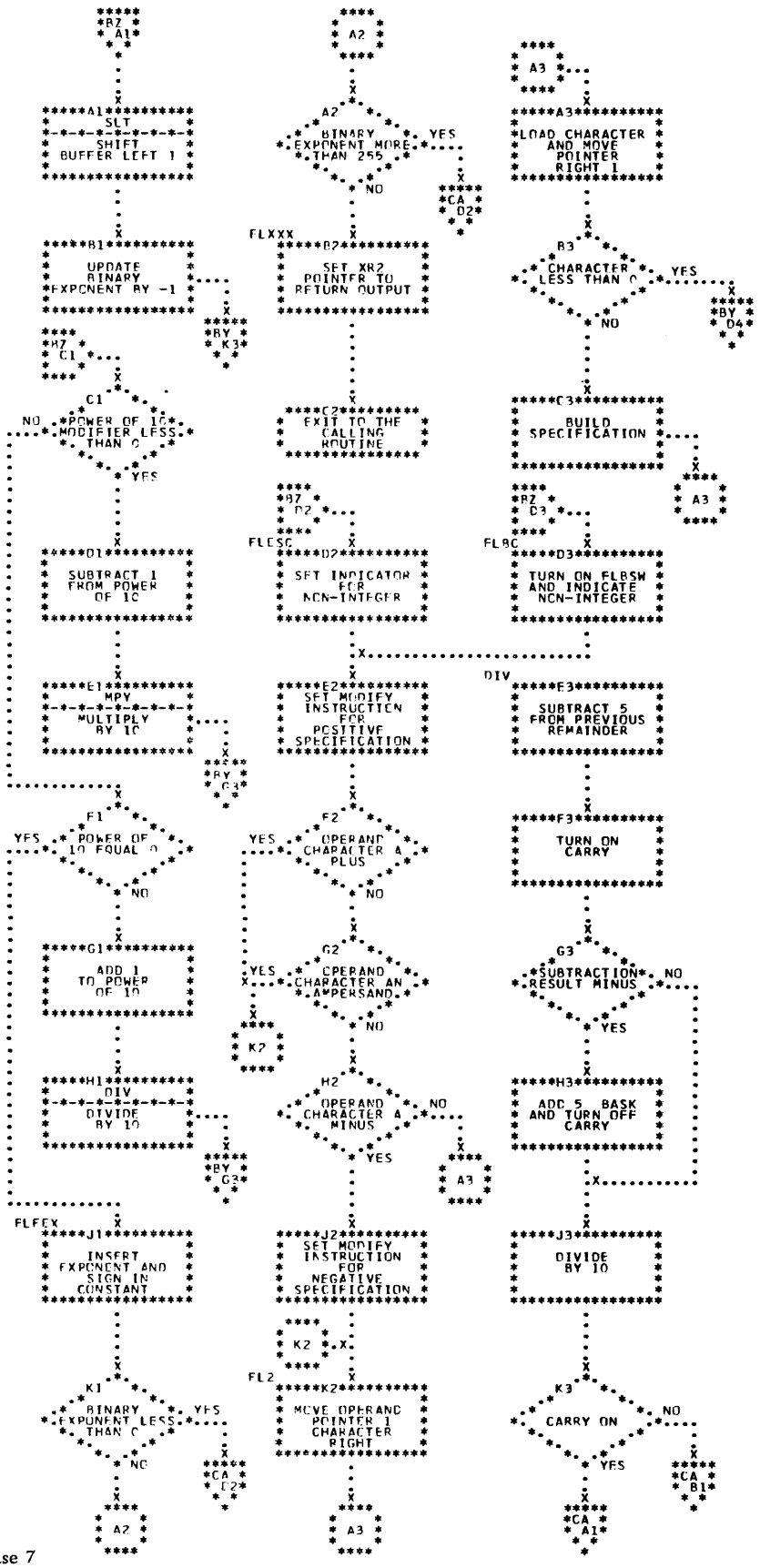


Chart BZ. The Assembler - Phase 7

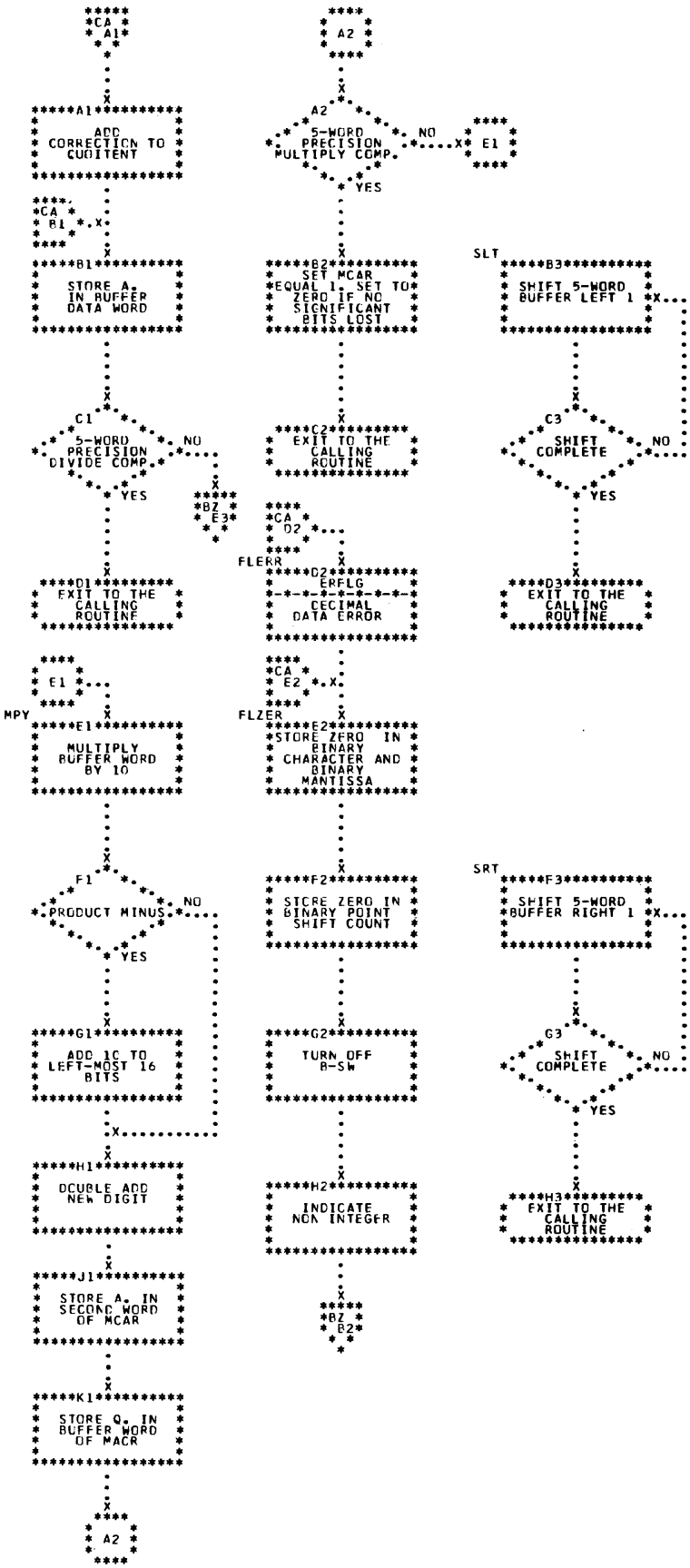


Chart CA. The Assembler - Phase 7

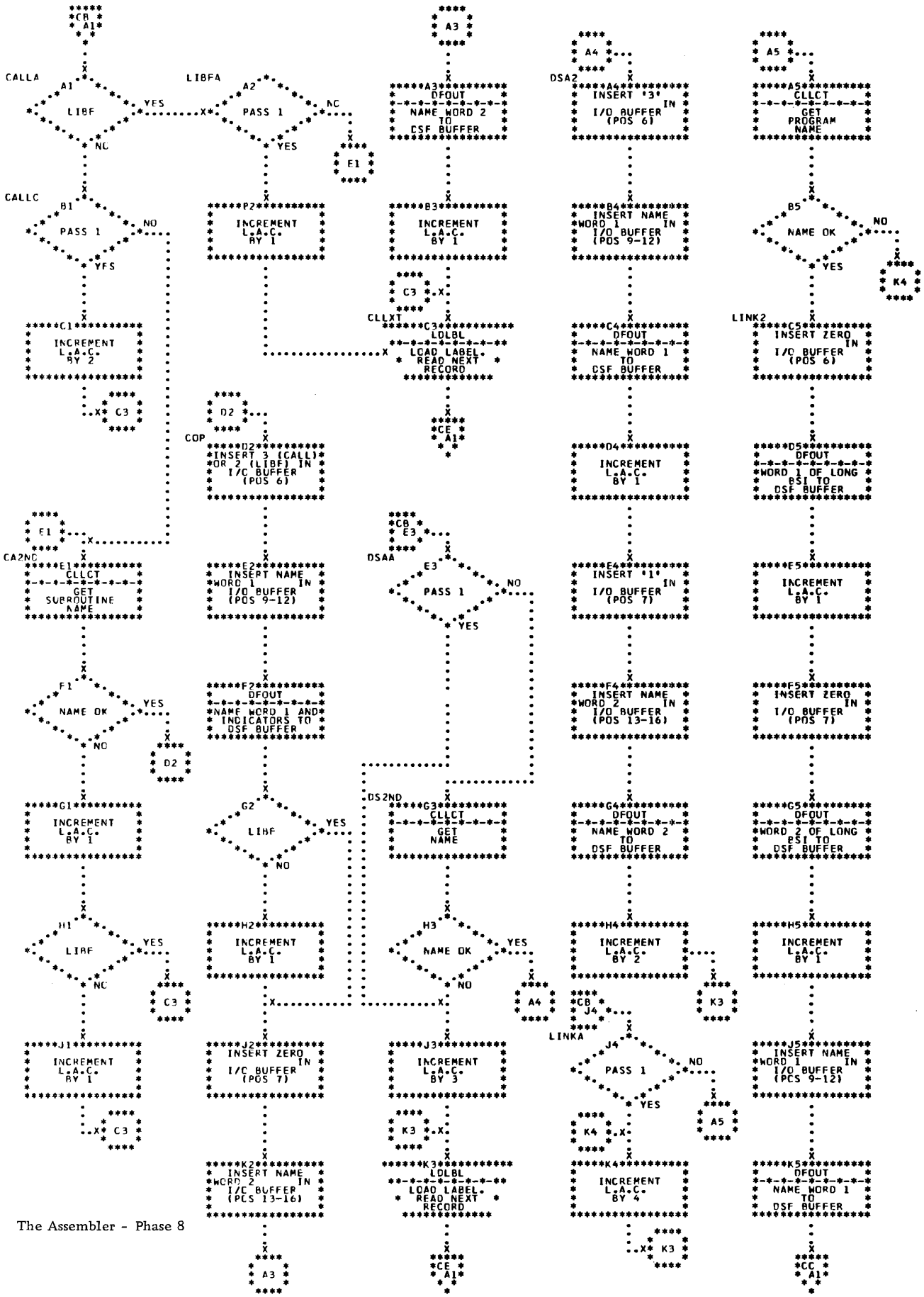


Chart CB. The Assembler - Phase 8

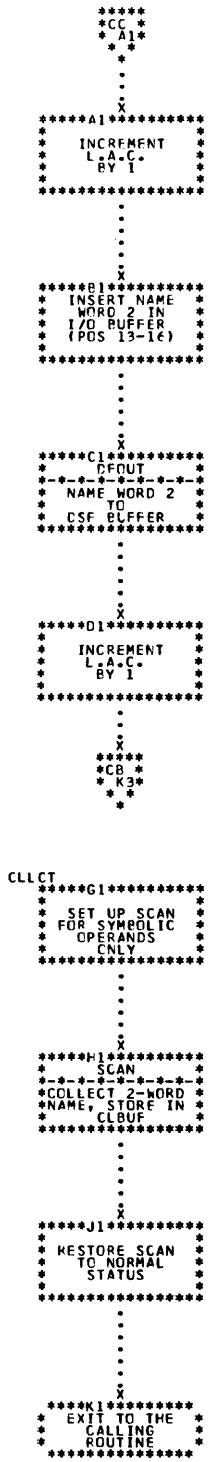


Chart CC. The Assembler - Phase 8

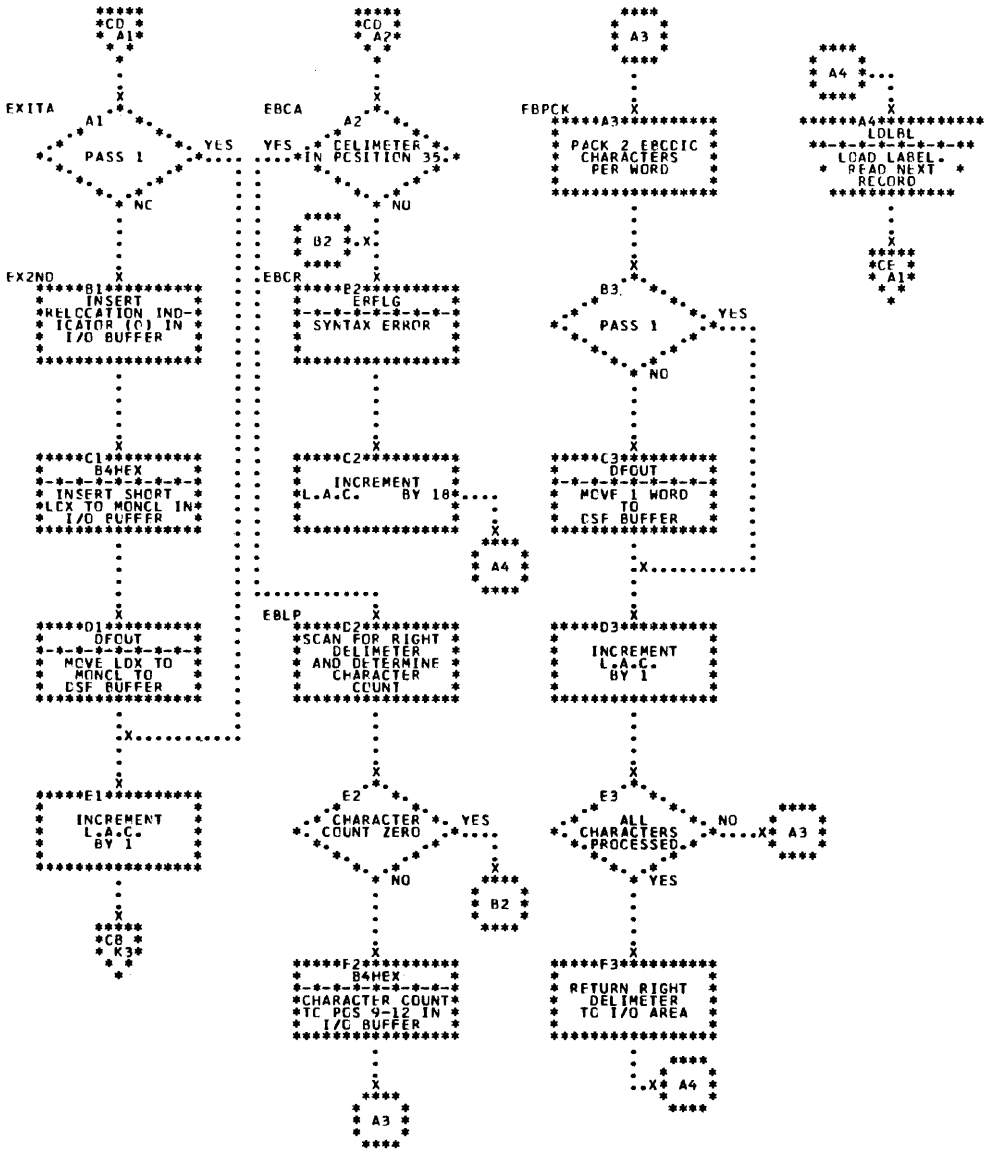


Chart CD. The Assembler - Phase 8

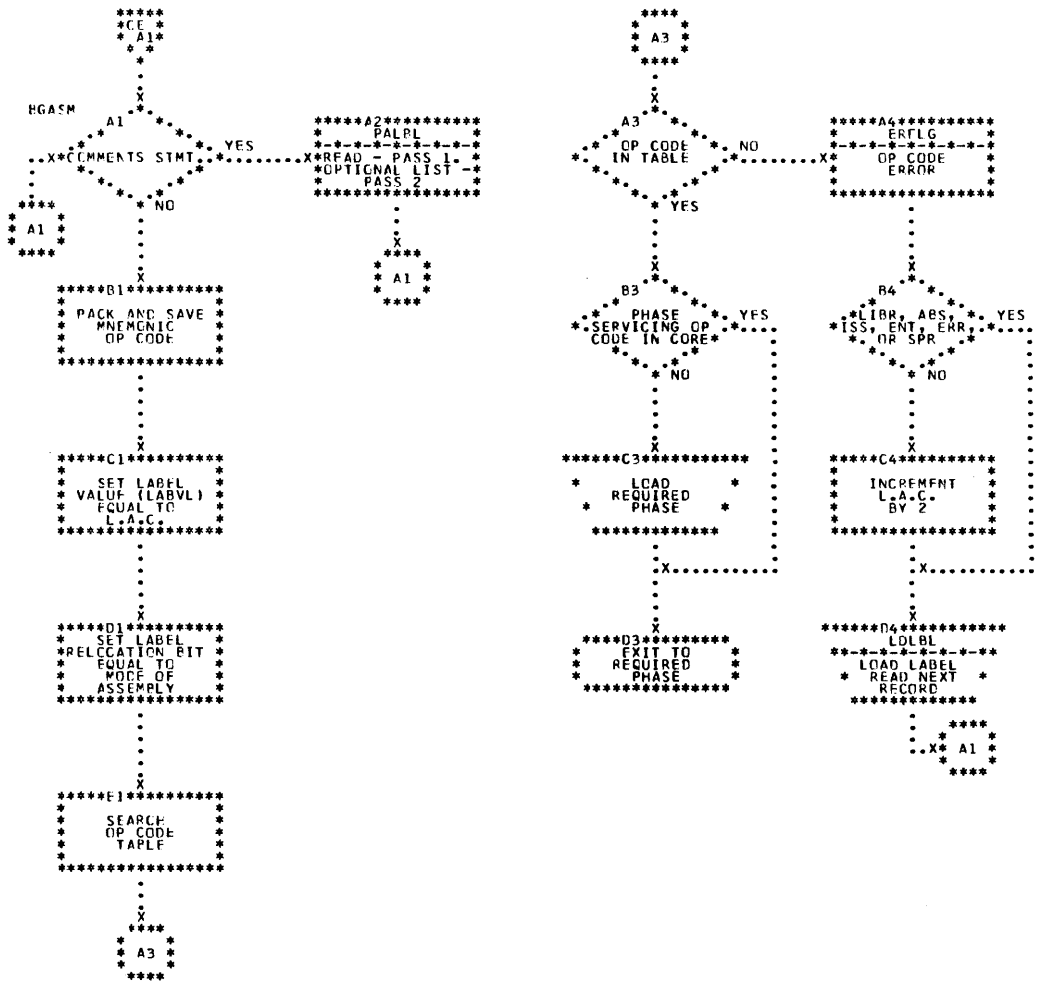


Chart CE. The Assembler - Phase 9

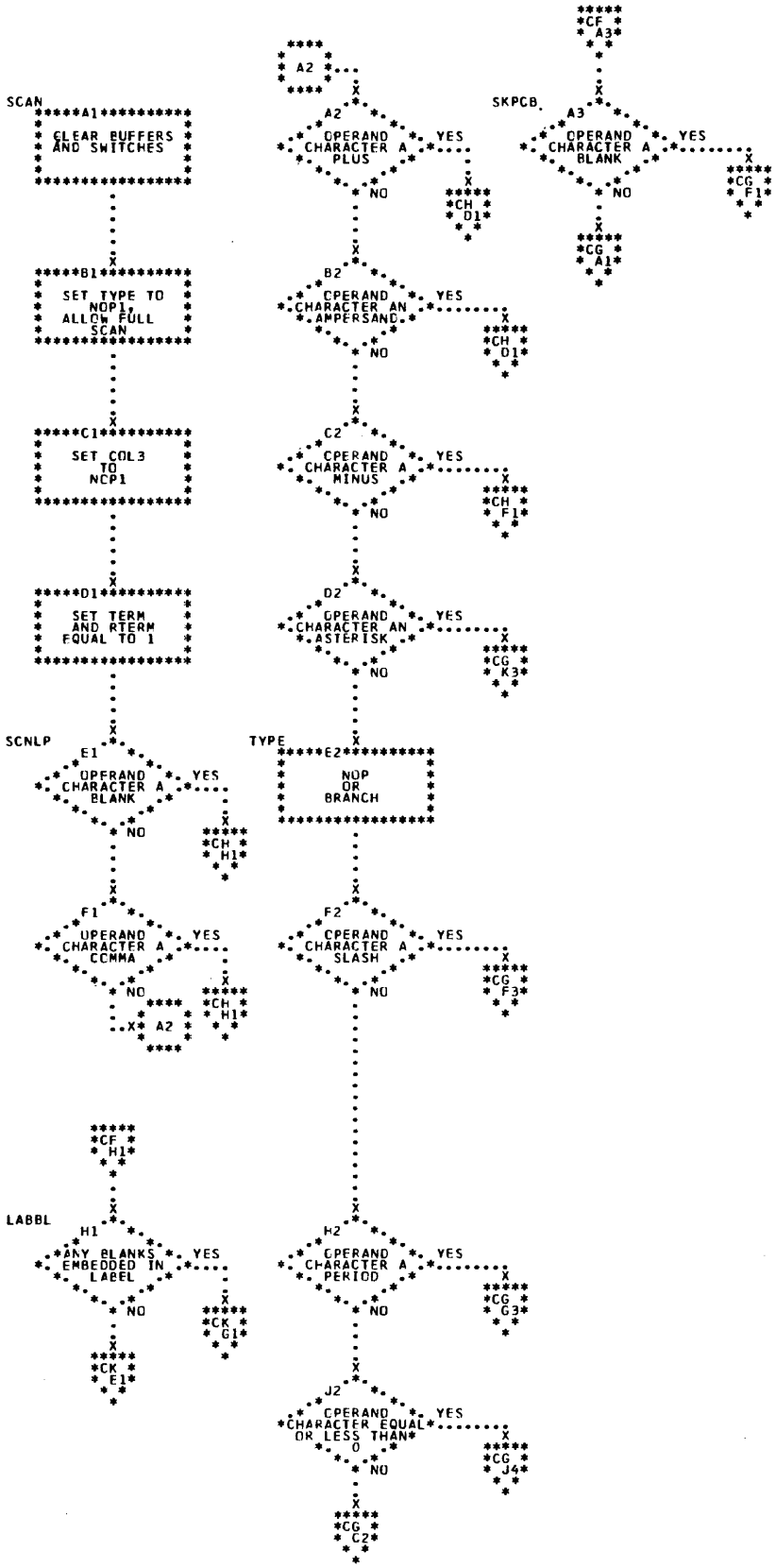


Chart CF. The Assembler - Phase 9

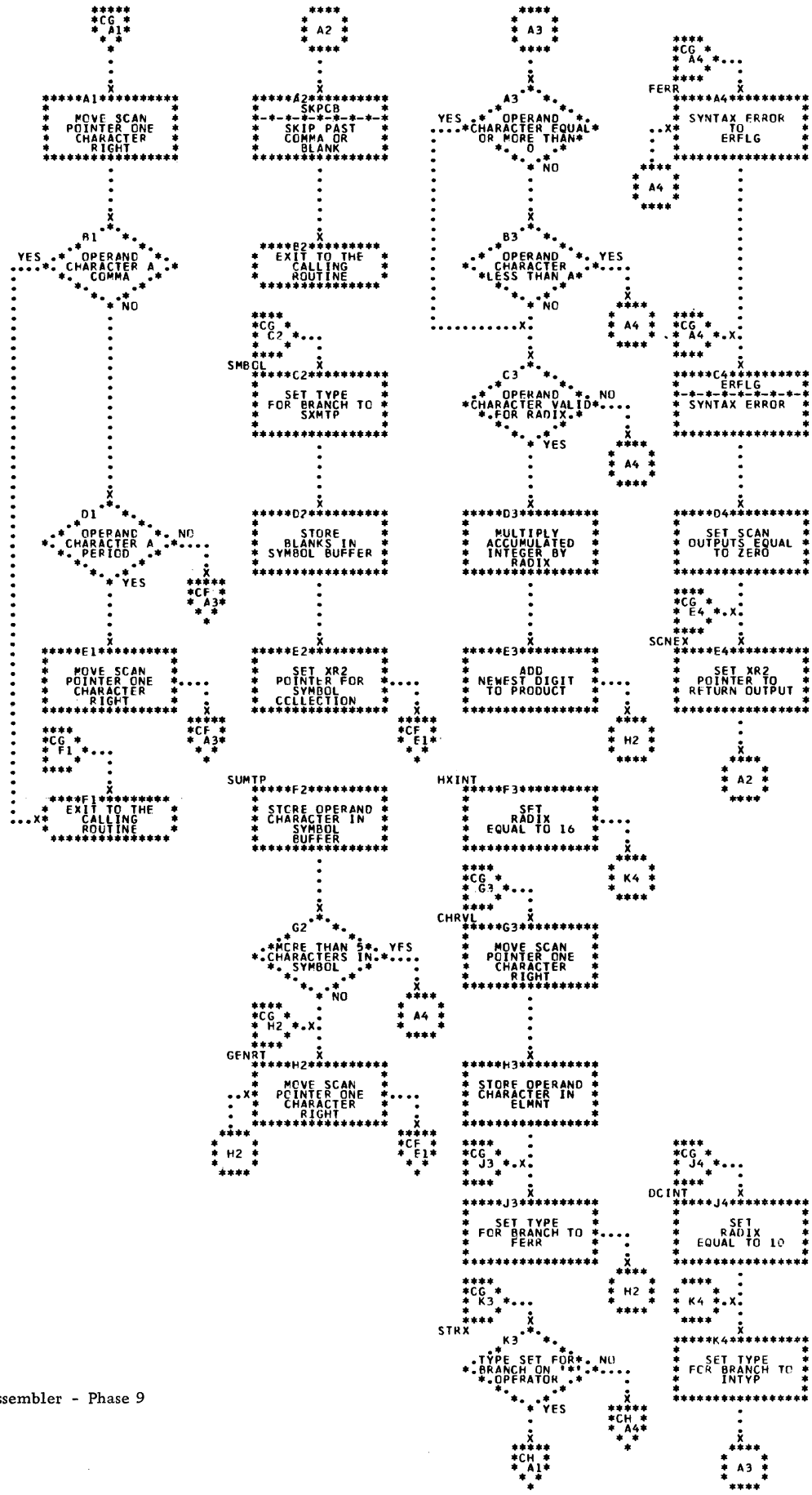


Chart CG. The Assembler - Phase 9

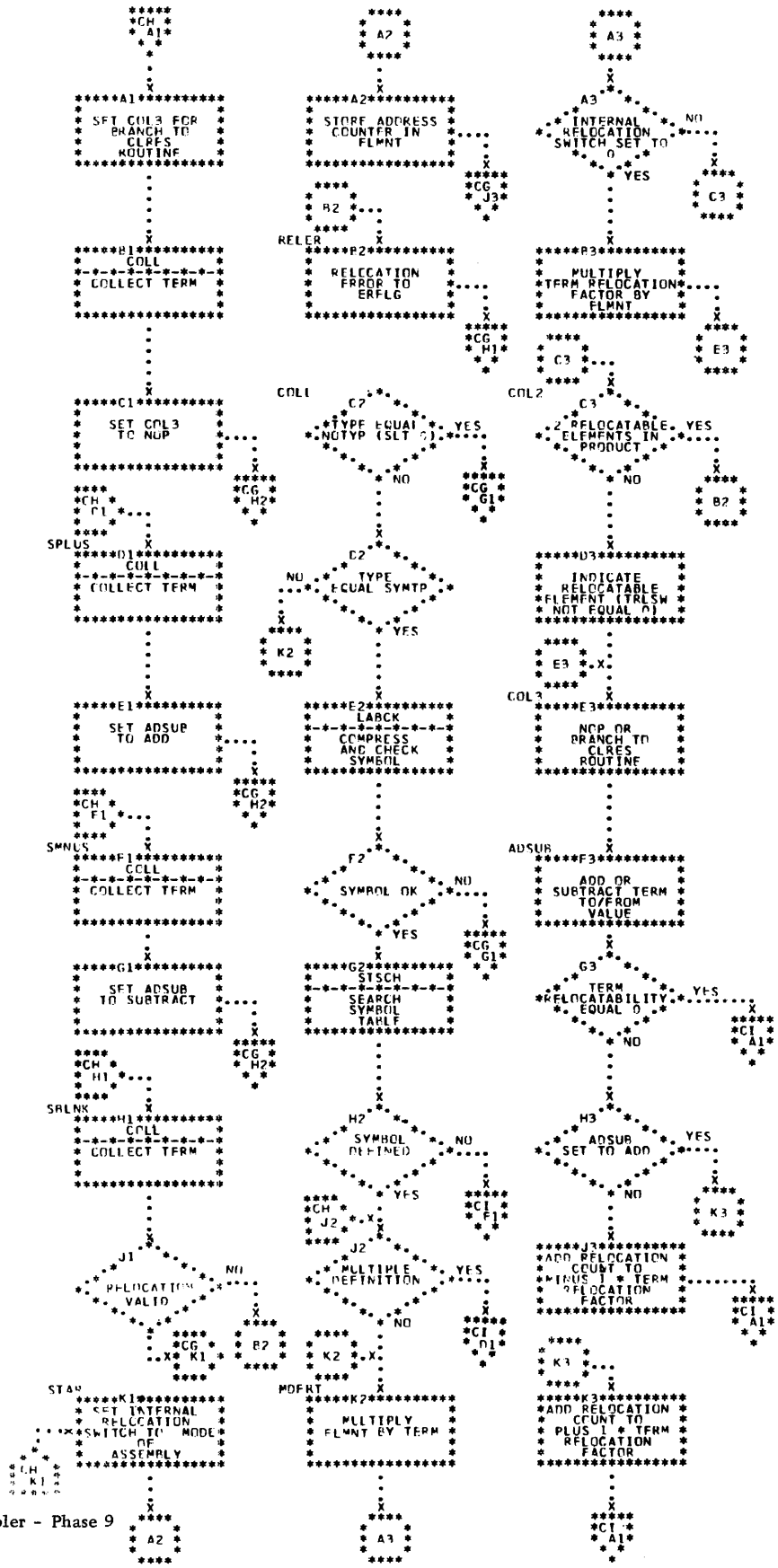


Chart CH. The Assembler - Phase 9

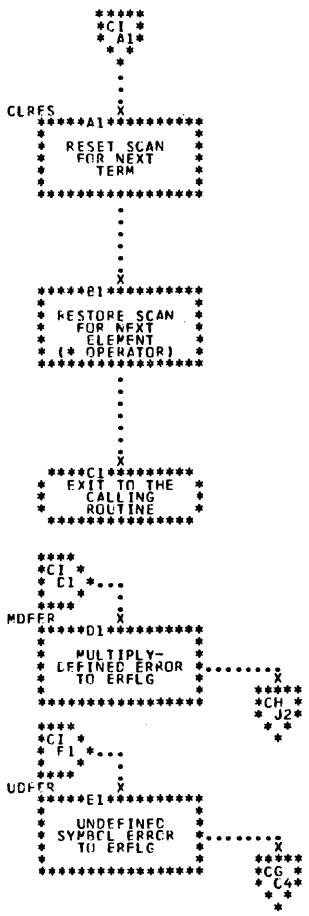


Chart CI. The Assembler - Phase 9

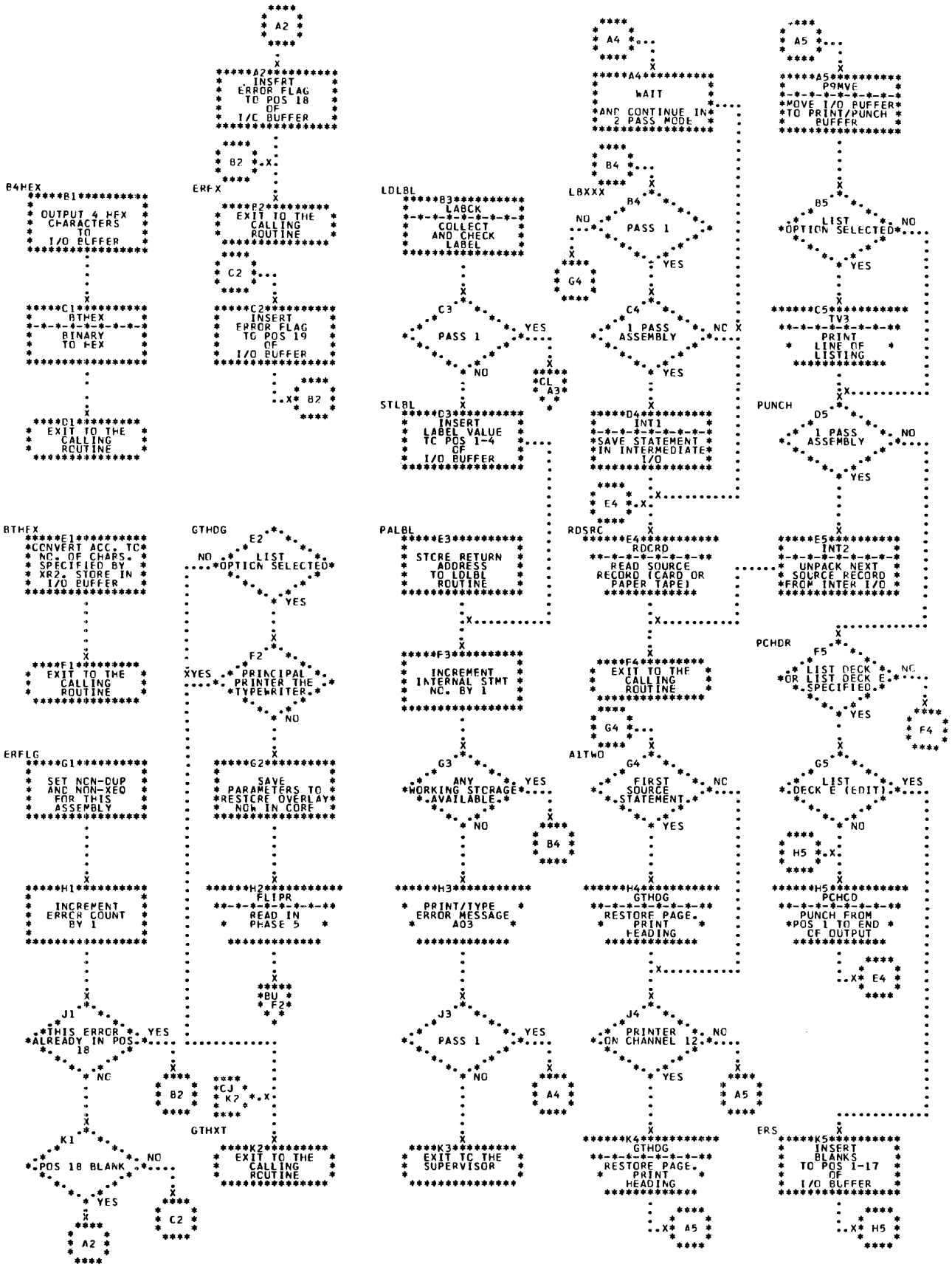


Chart CJ. The Assembler - Phase 9

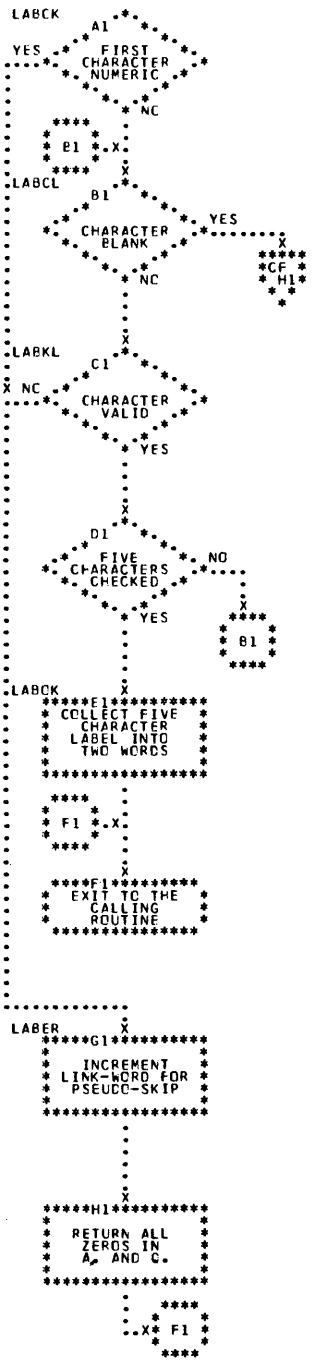


Chart CK. The Assembler - Phase 9

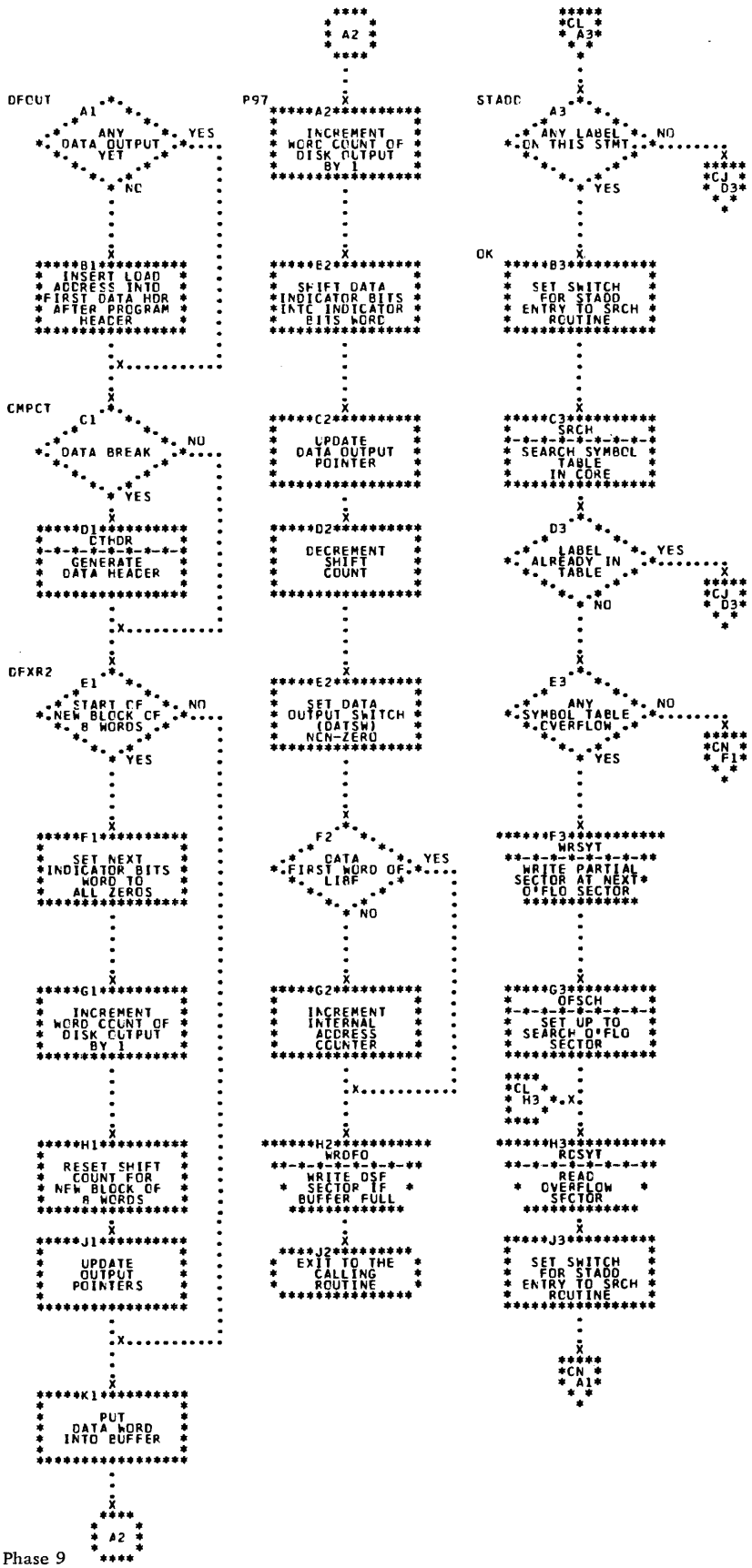


Chart CL. The Assembler - Phase 9

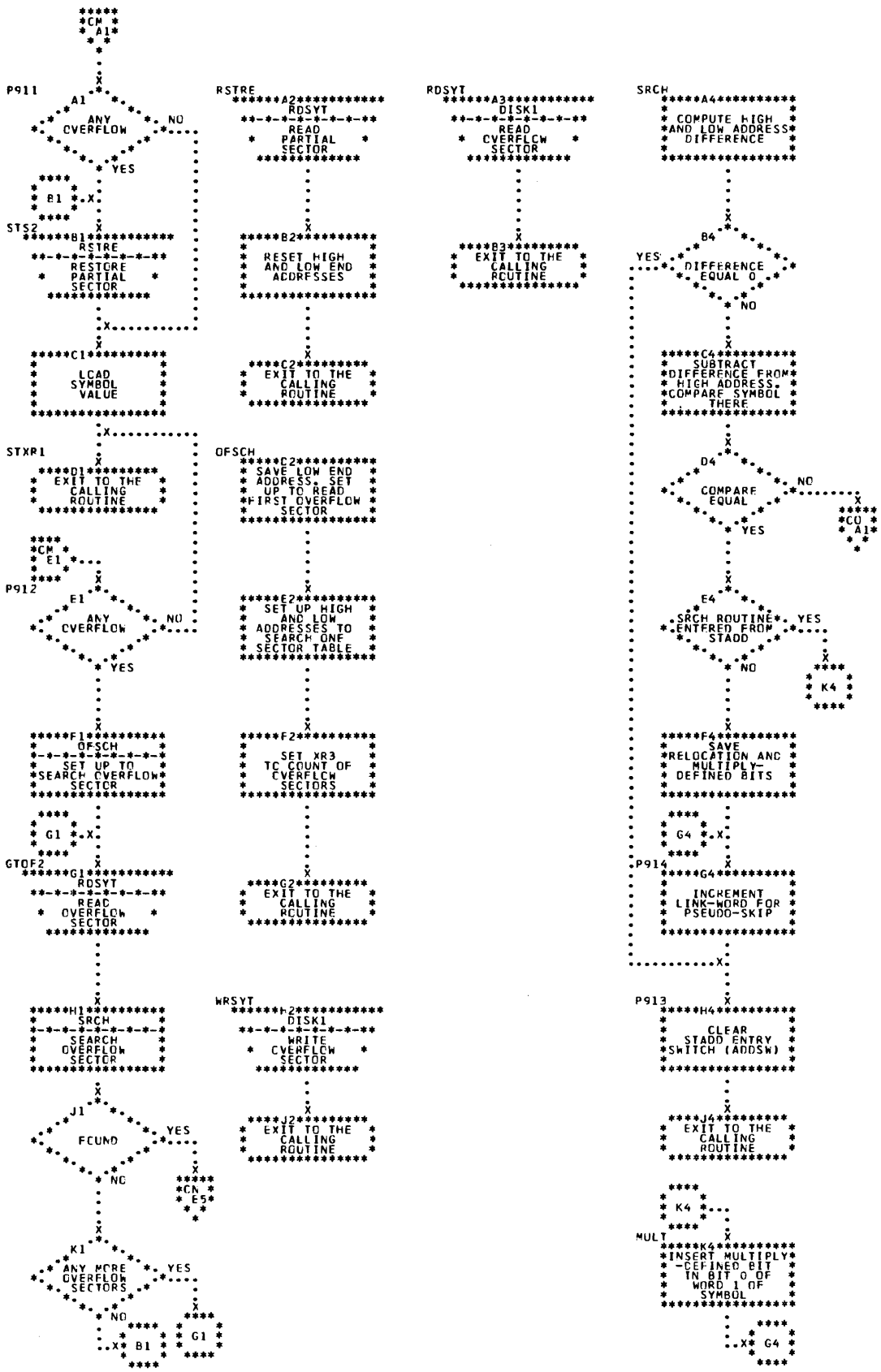


Chart CM. The Assembler - Phase 9

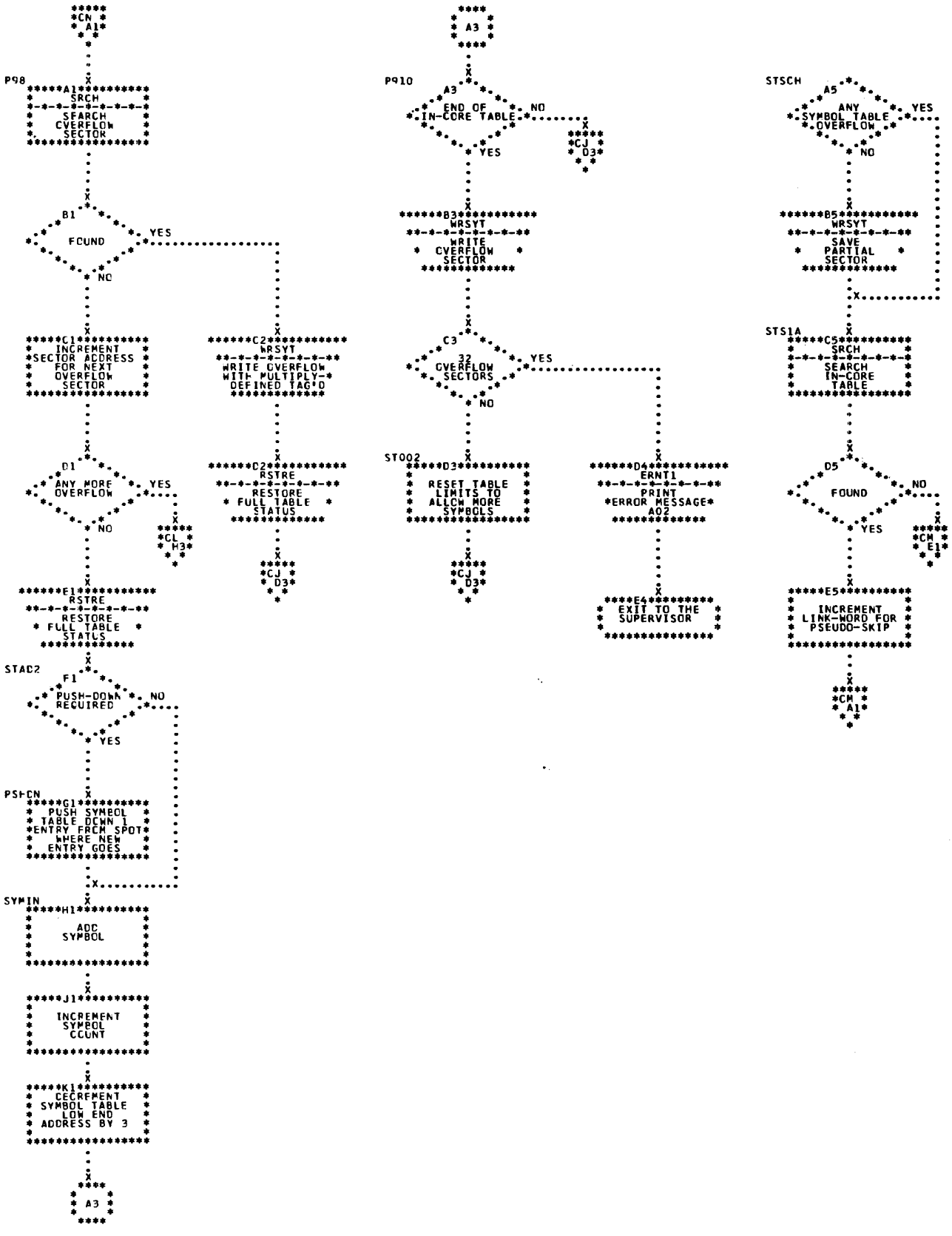


Chart CN. The Assembler - Phase 9

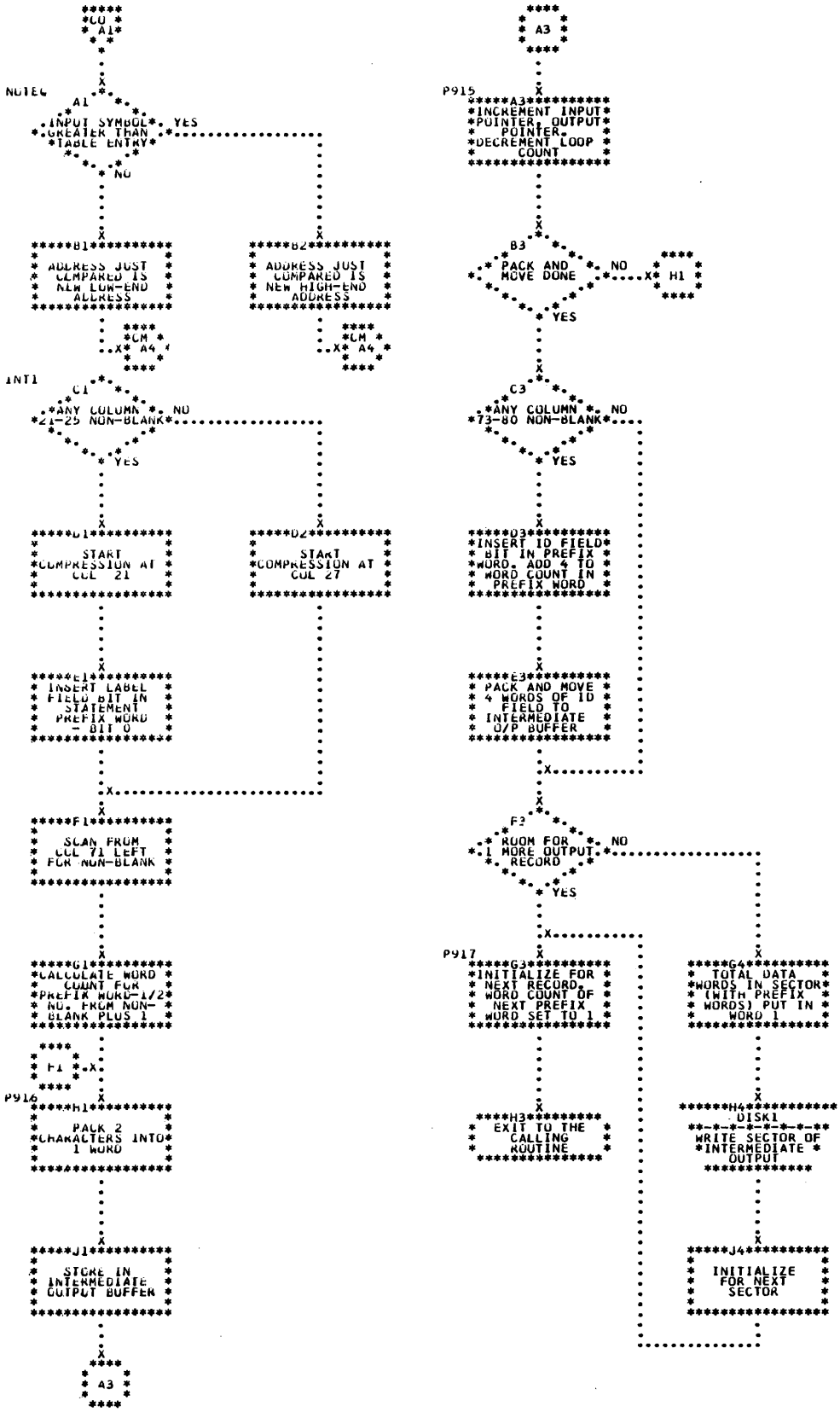


Chart CO. The Assembler - Phase 9

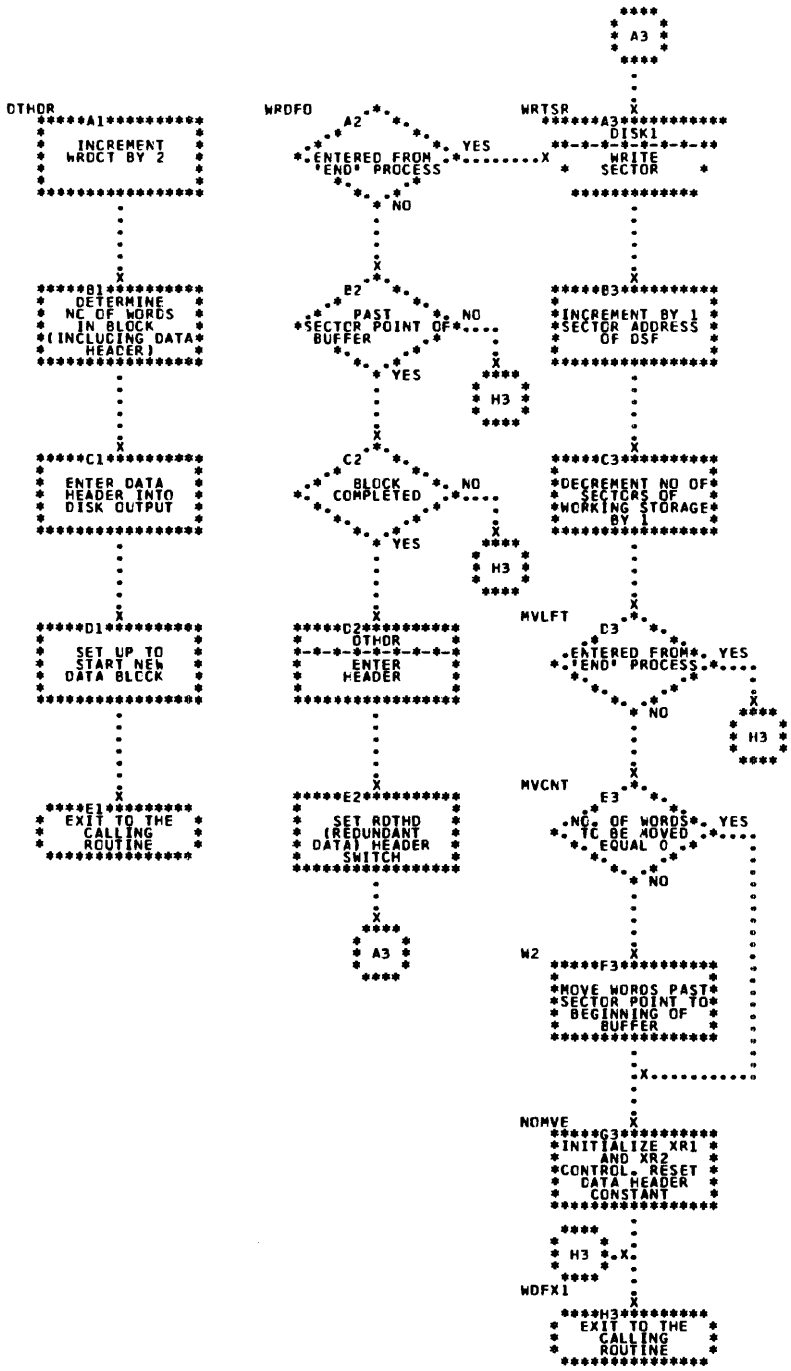


Chart CP. The Assembler - Phase 10

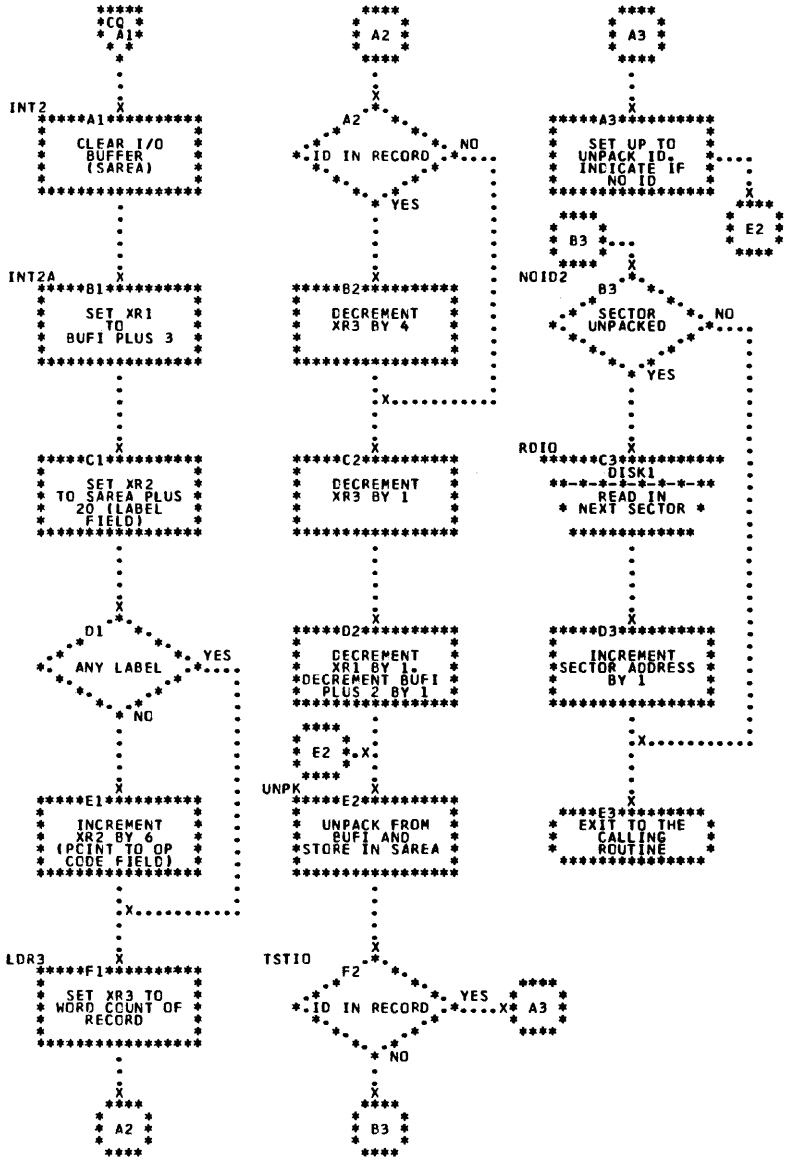


Chart CQ. The Assembler - Phase 11

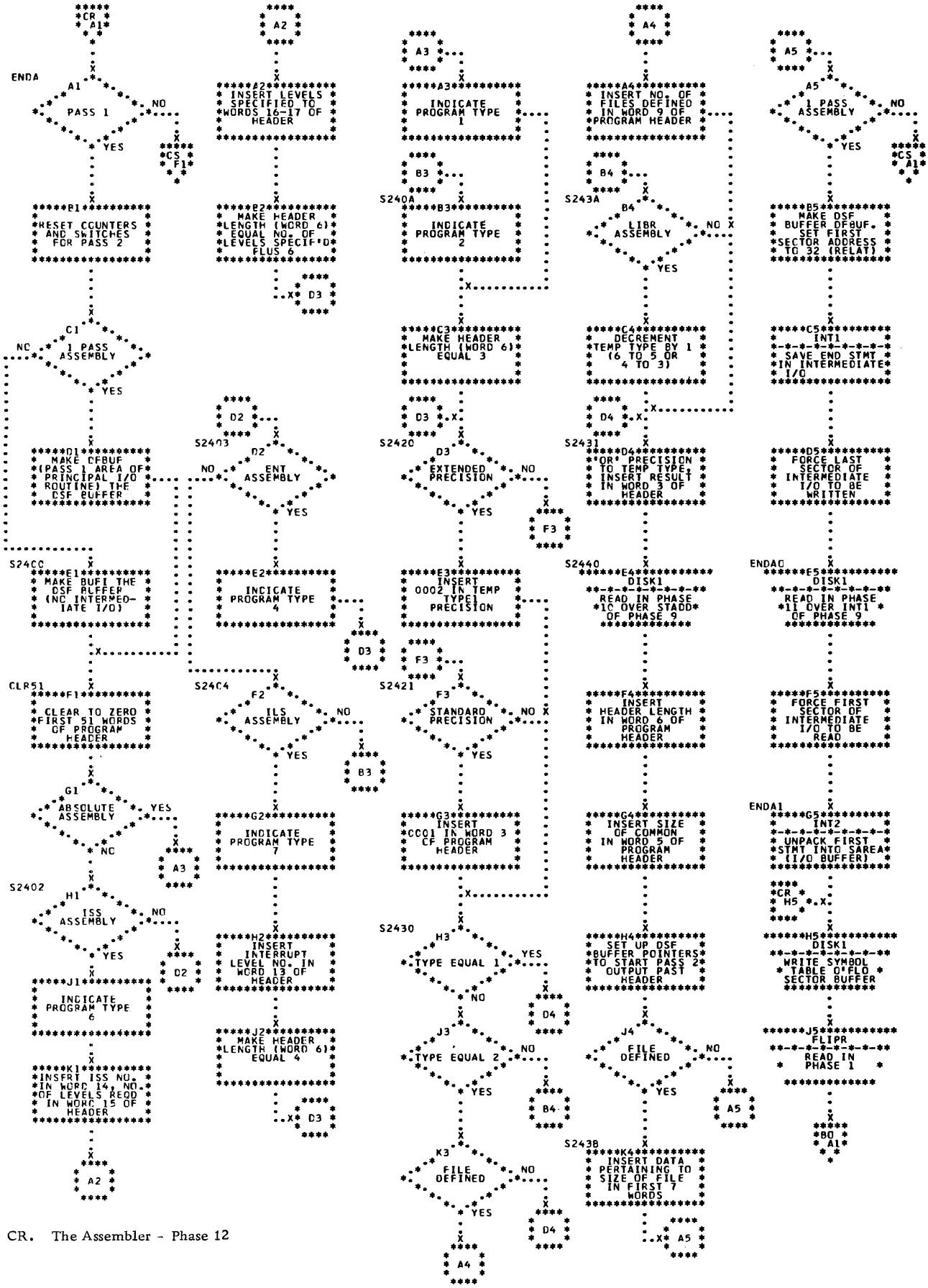


Chart CR. The Assembler - Phase 12

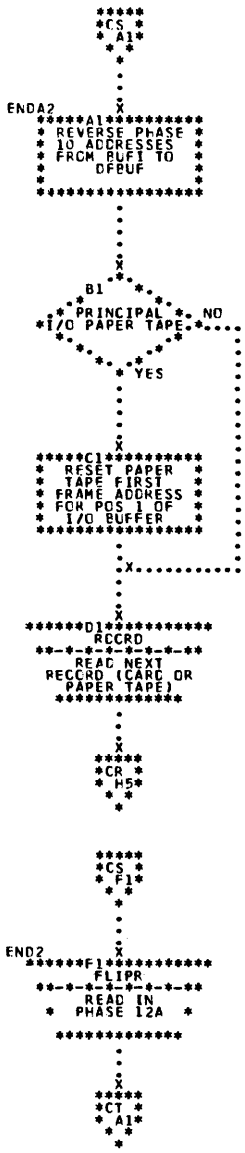


Chart CS. The Assembler - Phase 12

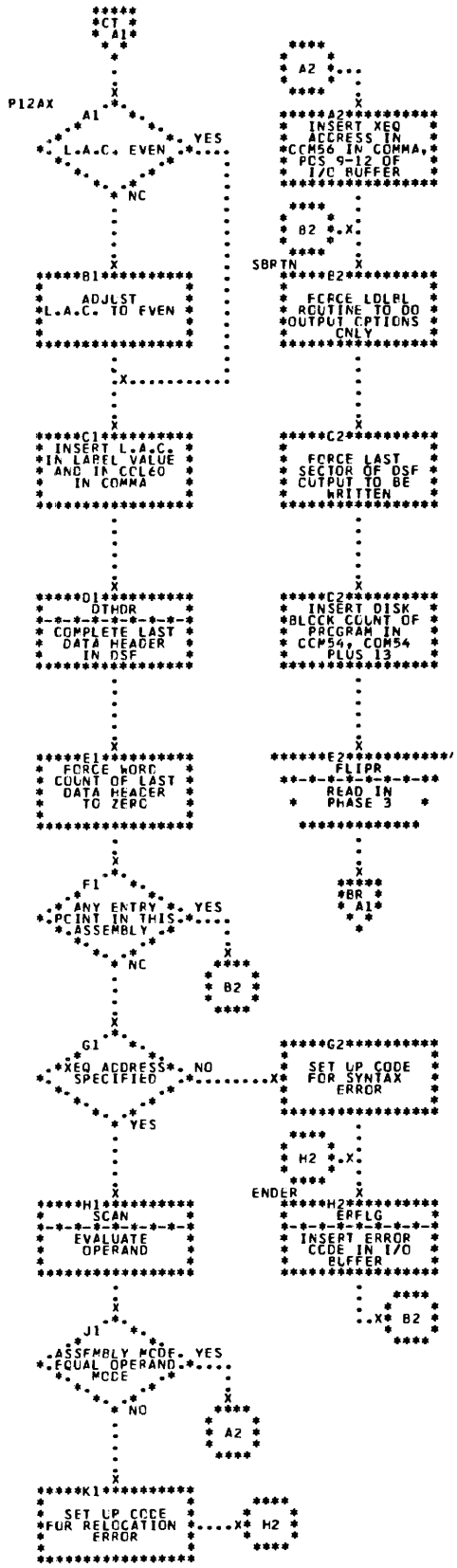


Chart CT. The Assembler - Phase 12A

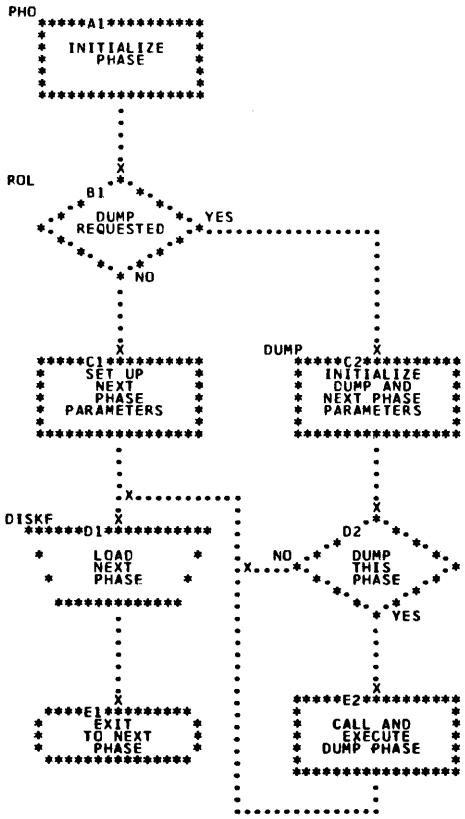


Chart DA. FORTRAN - Phase 1

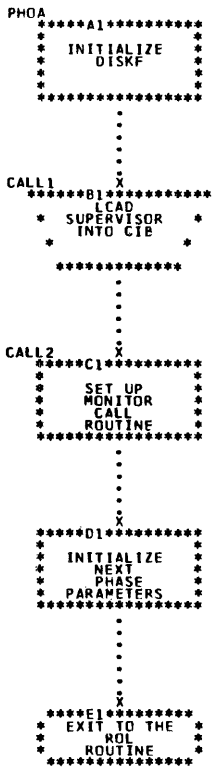


Chart DB. FORTRAN - Phase 2

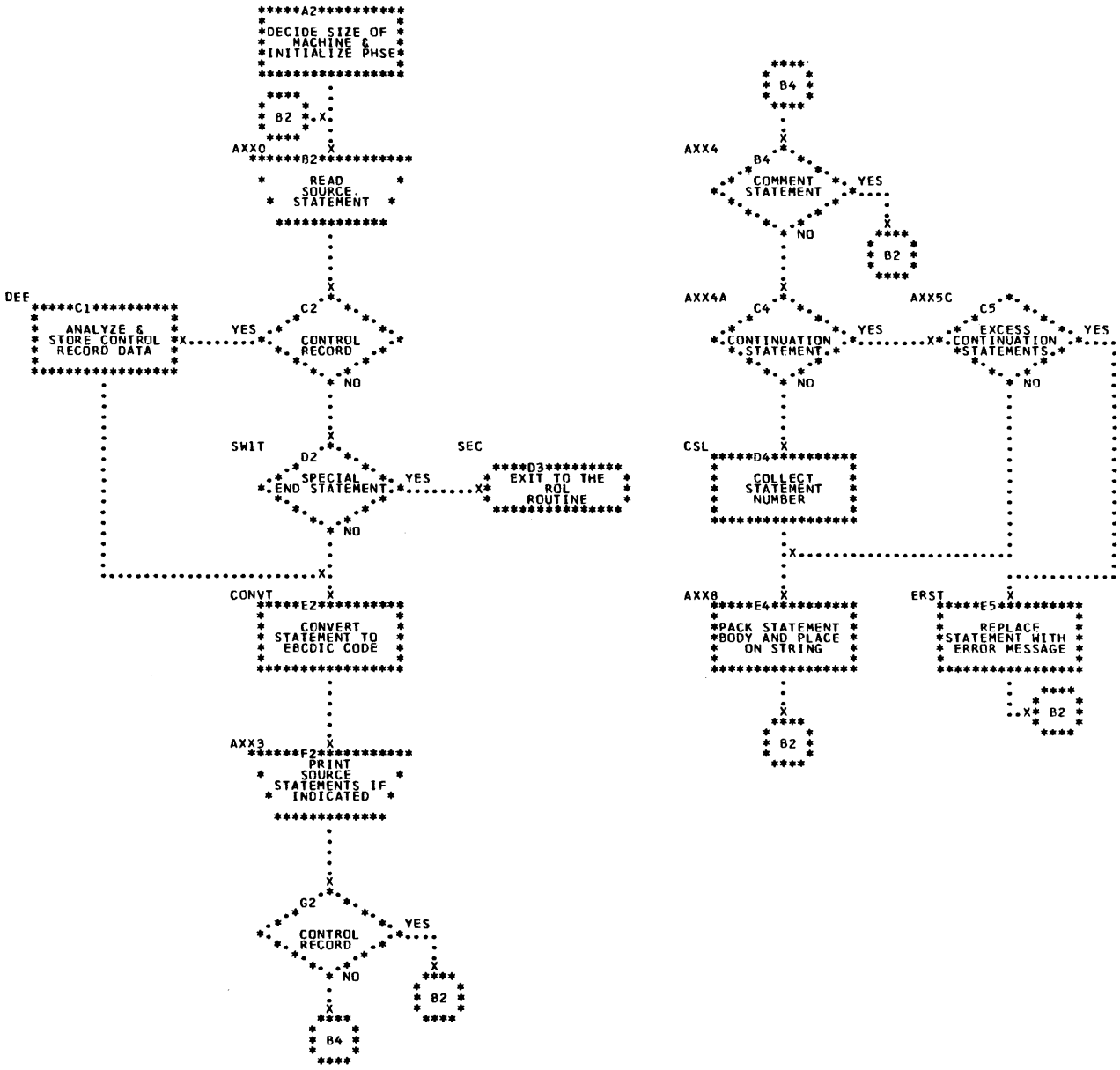


Chart DC. FORTRAN - Phase 3

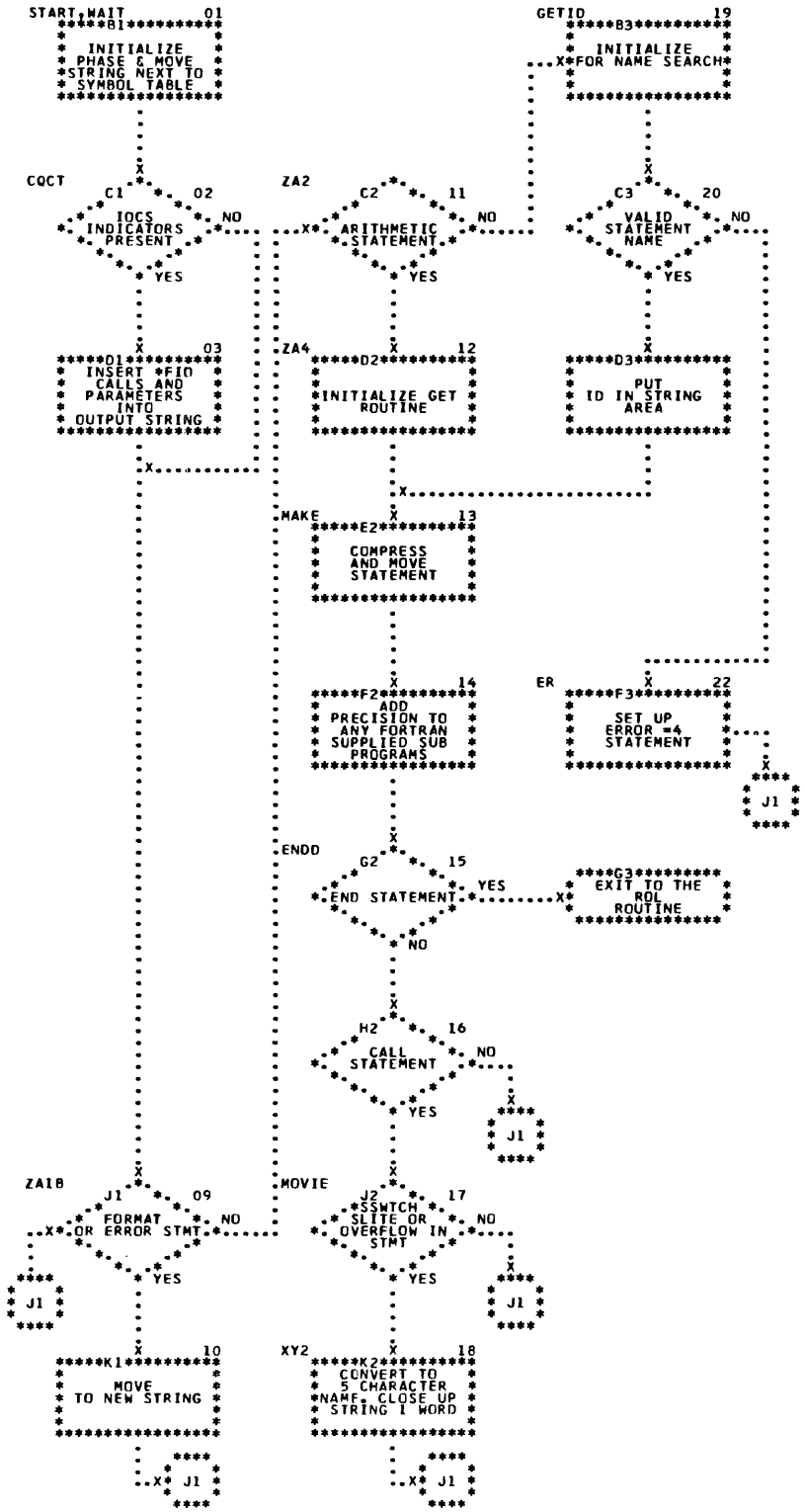


Chart DD. FORTRAN - Phase 4

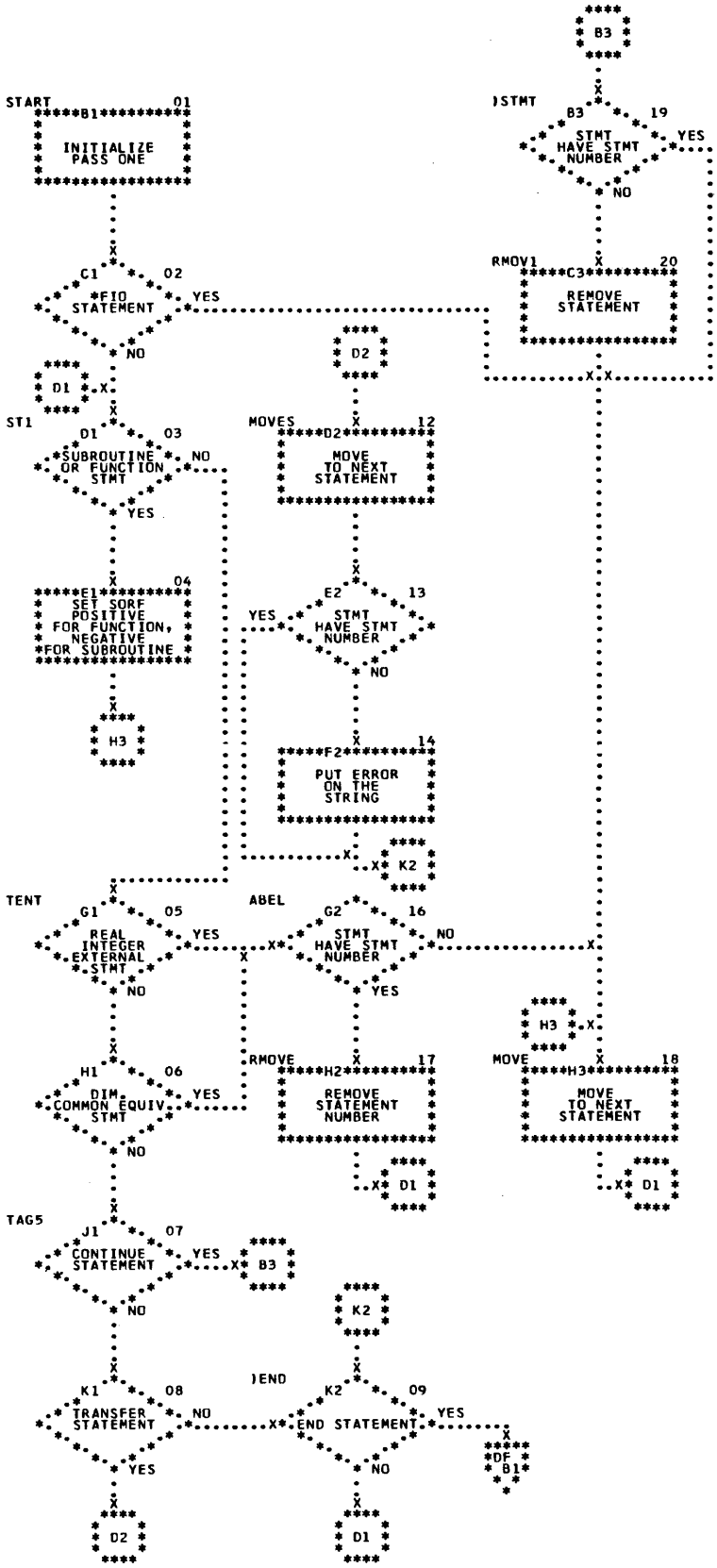


Chart DE. FORTRAN - Phase 5

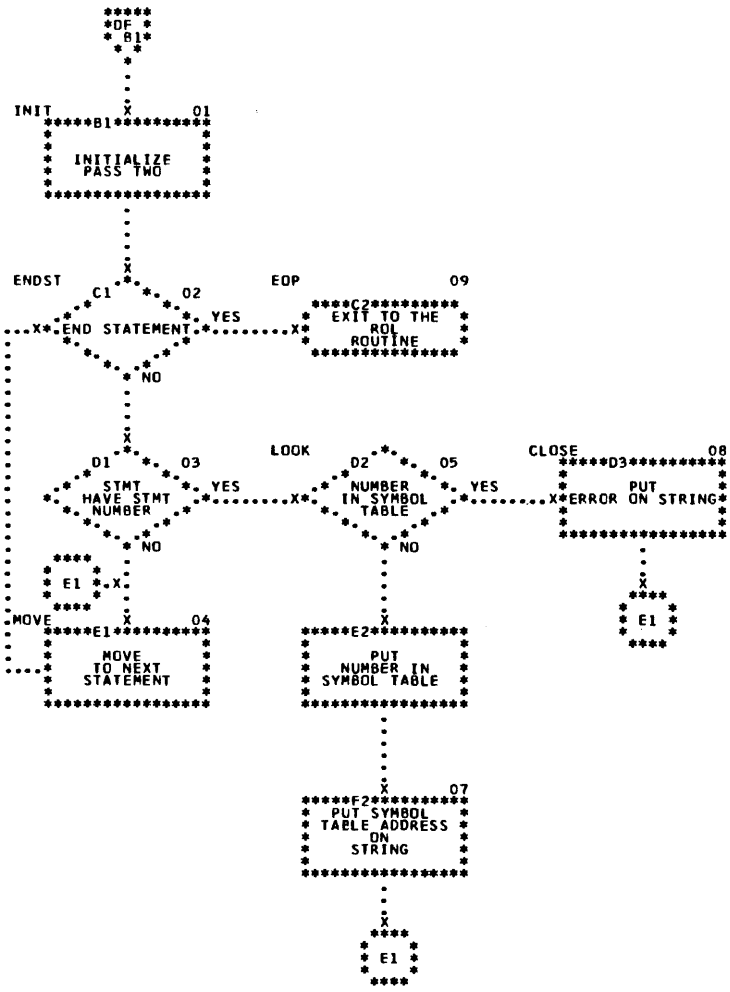


Chart DF. FORTRAN - Phase 5

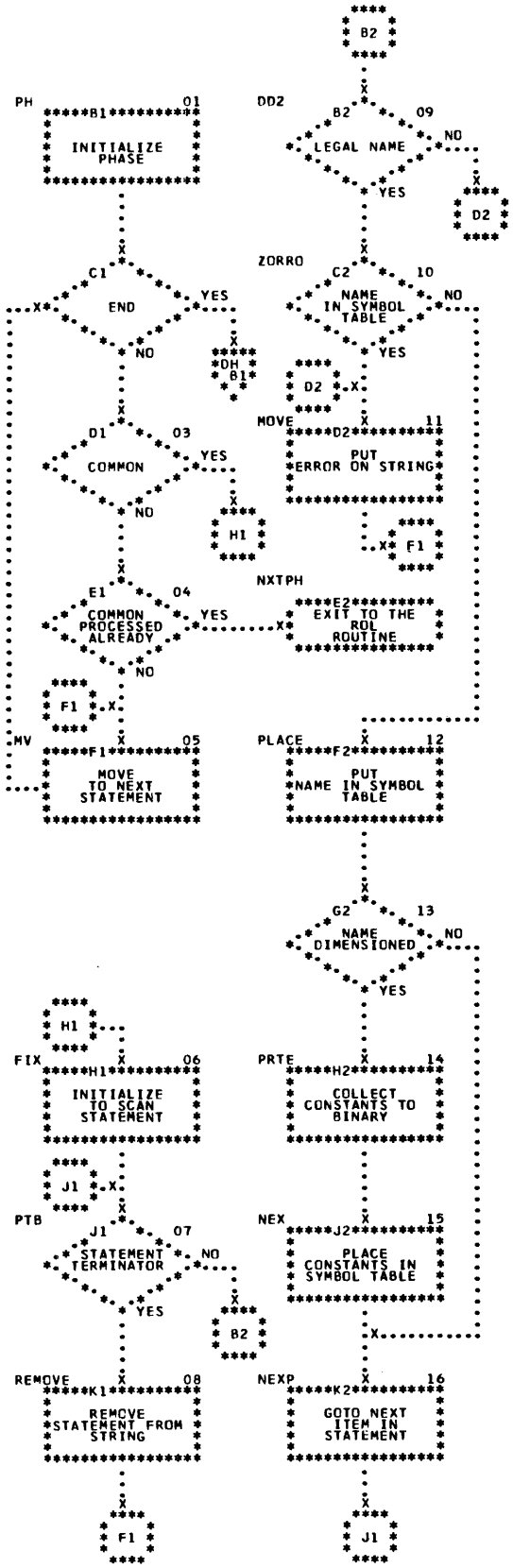


Chart DG. FORTRAN - Phase 6

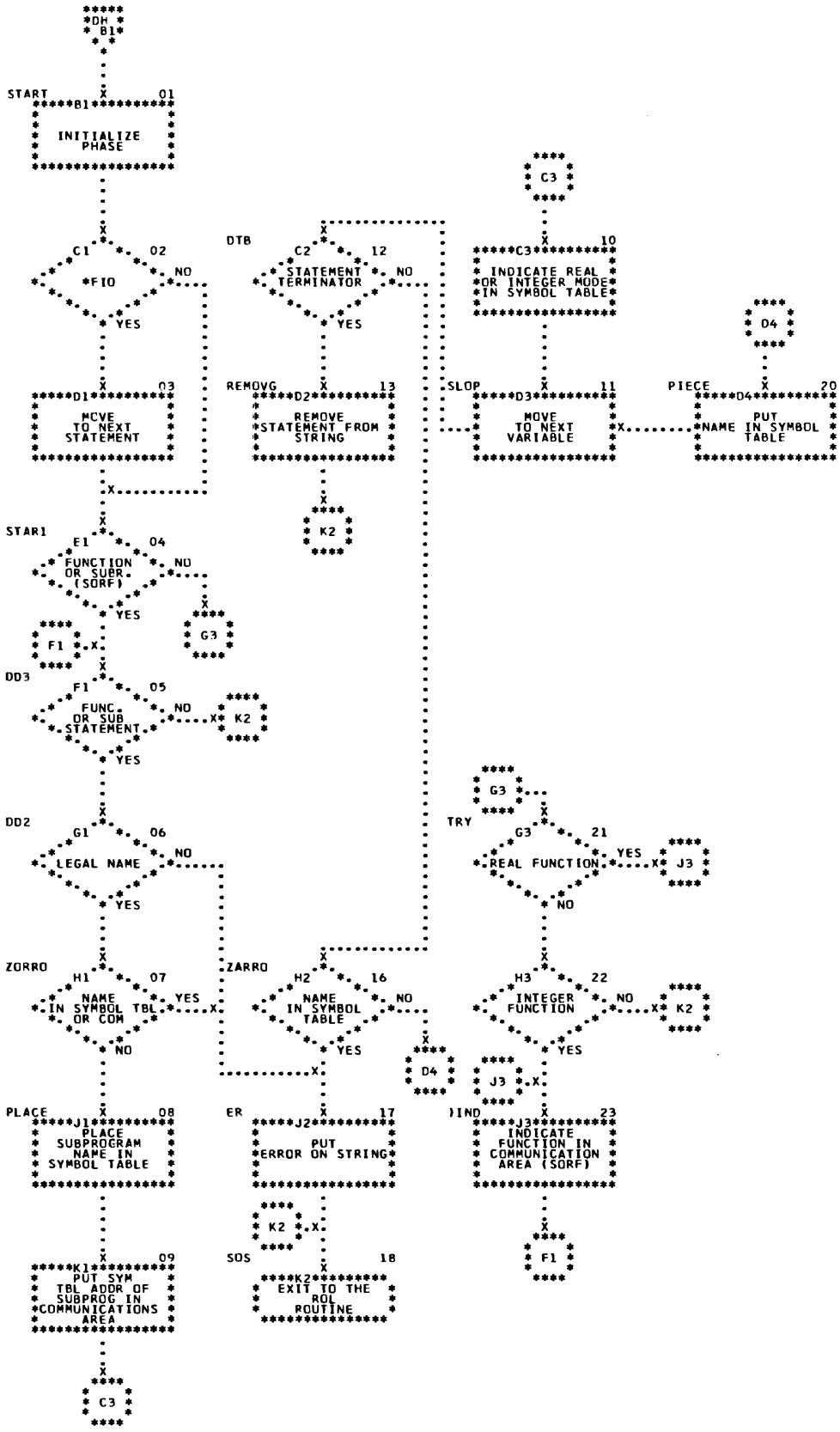


Chart DH. FORTRAN - Phase 6

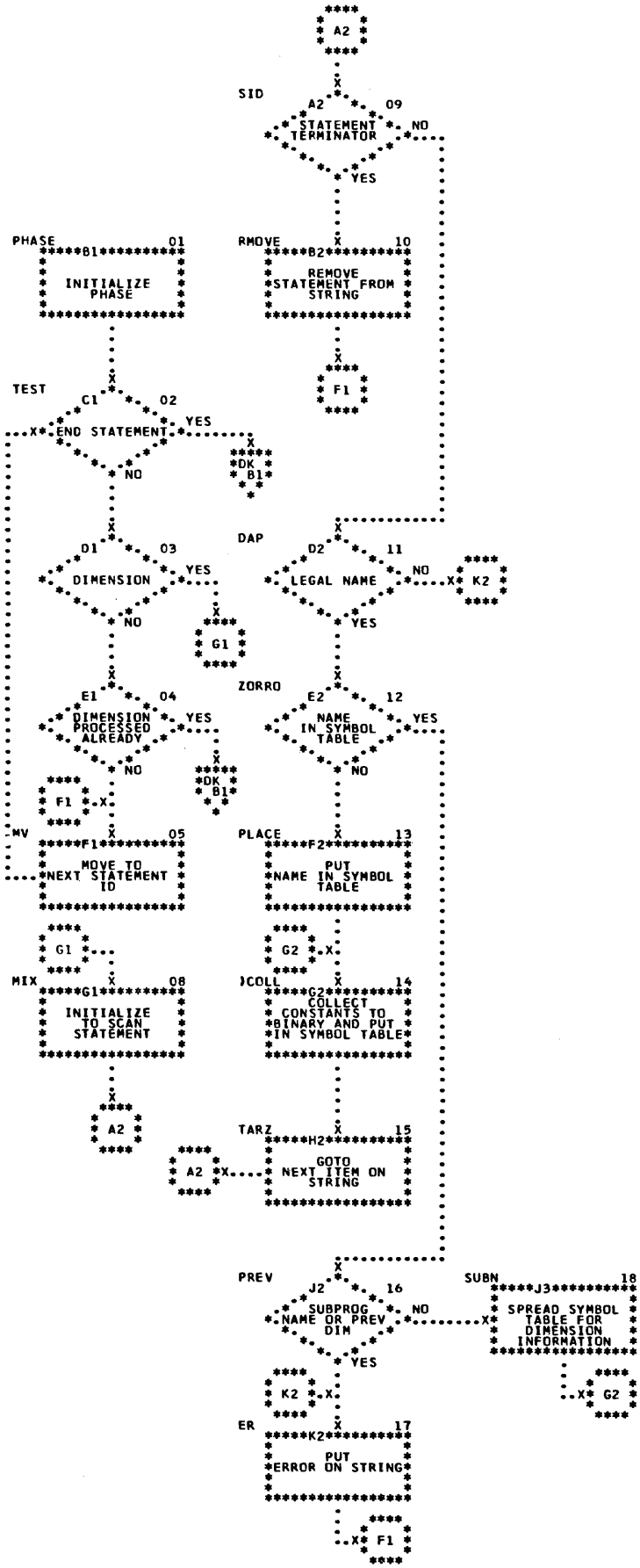


Chart DJ. FORTRAN - Phase 7

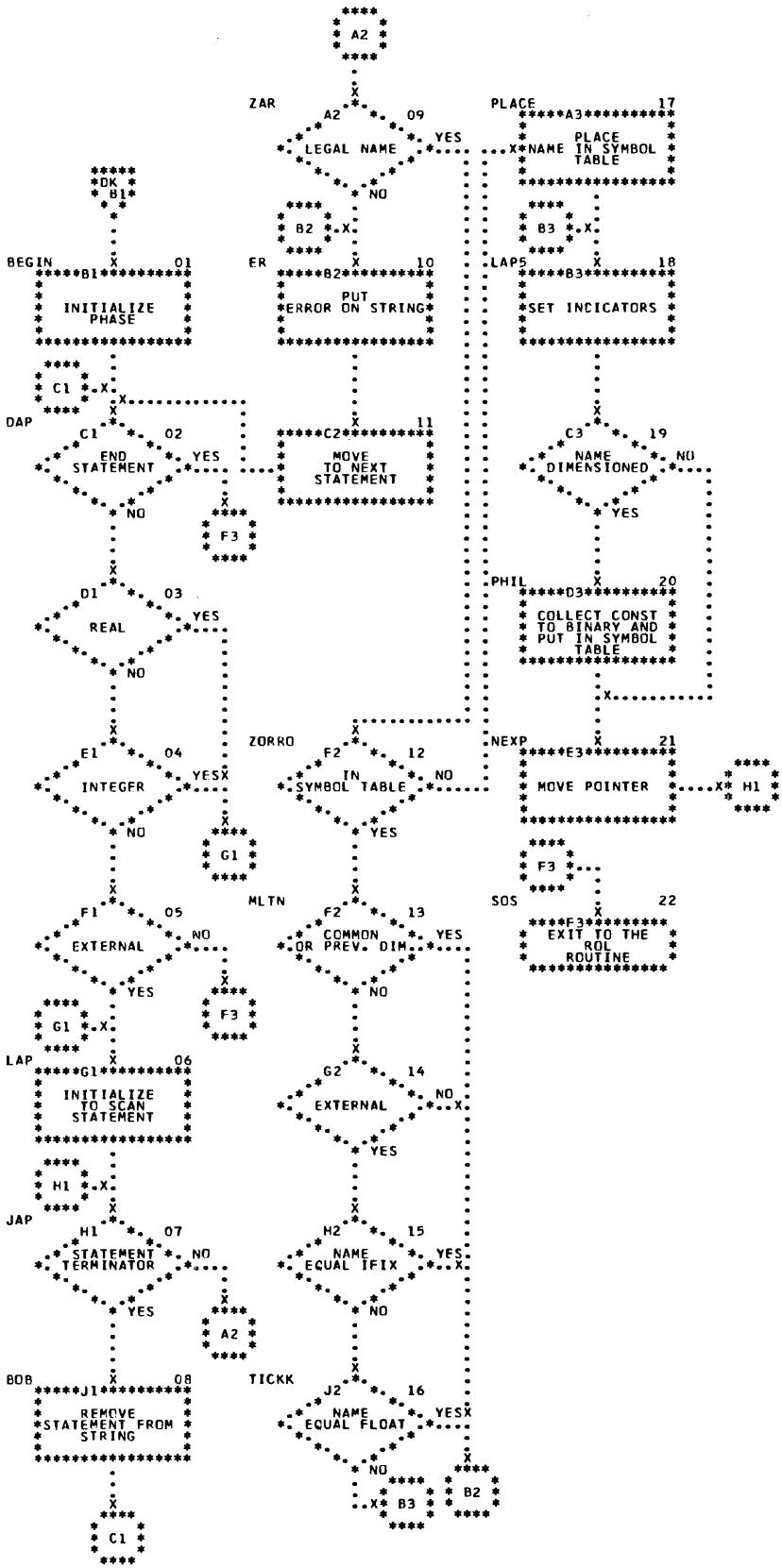


Chart DK. FORTRAN - Phase 7

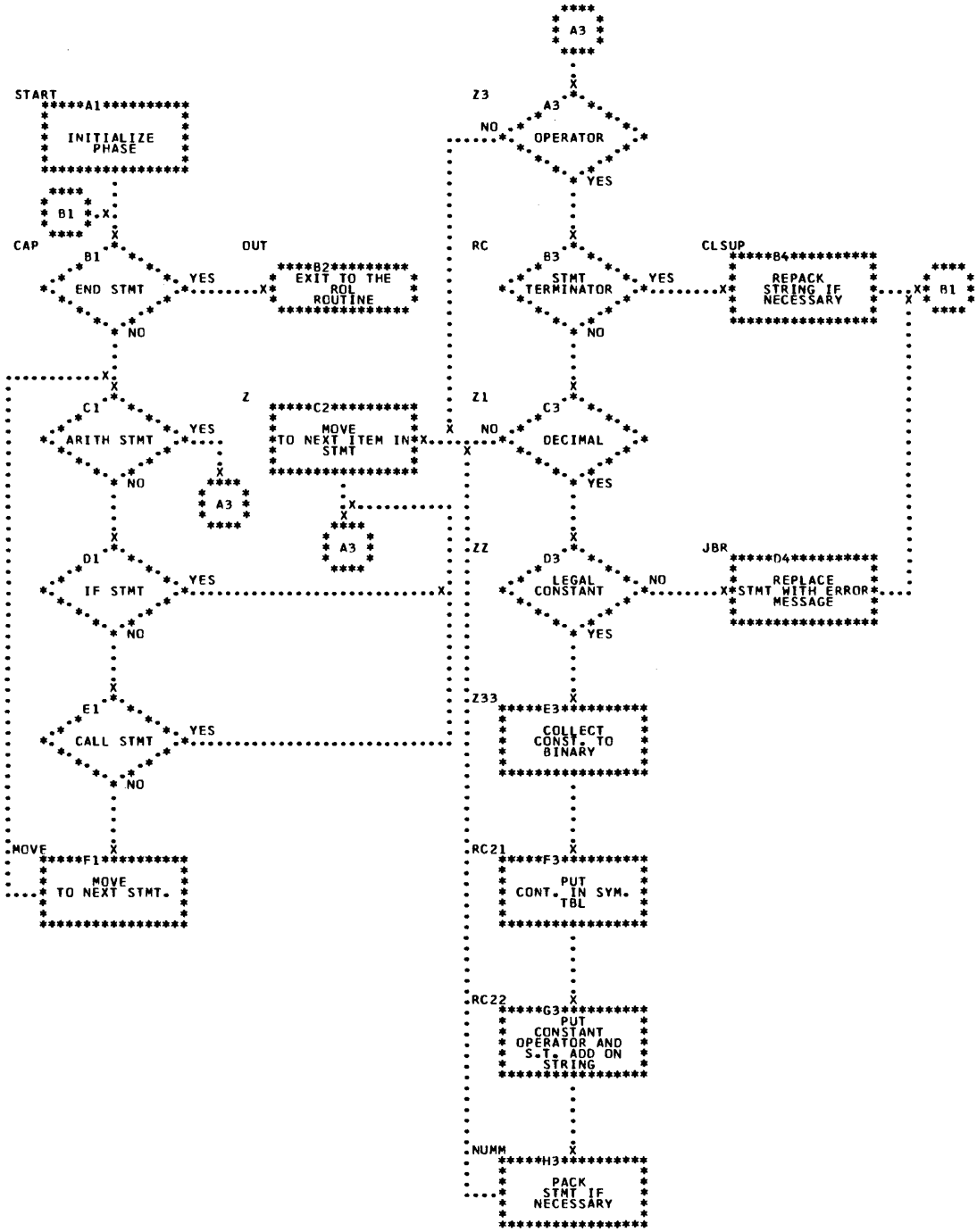


Chart DL. FORTRAN - Phase 8

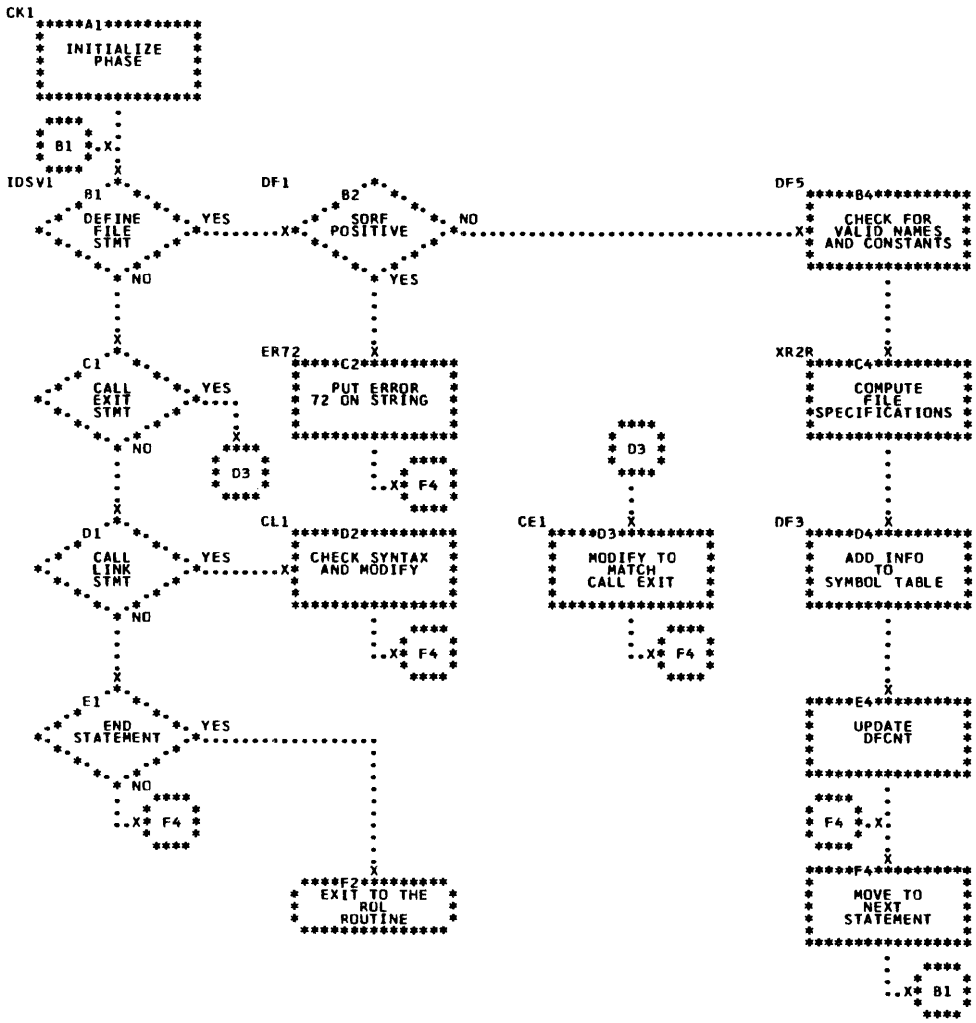


Chart DM. FORTRAN - Phase 9

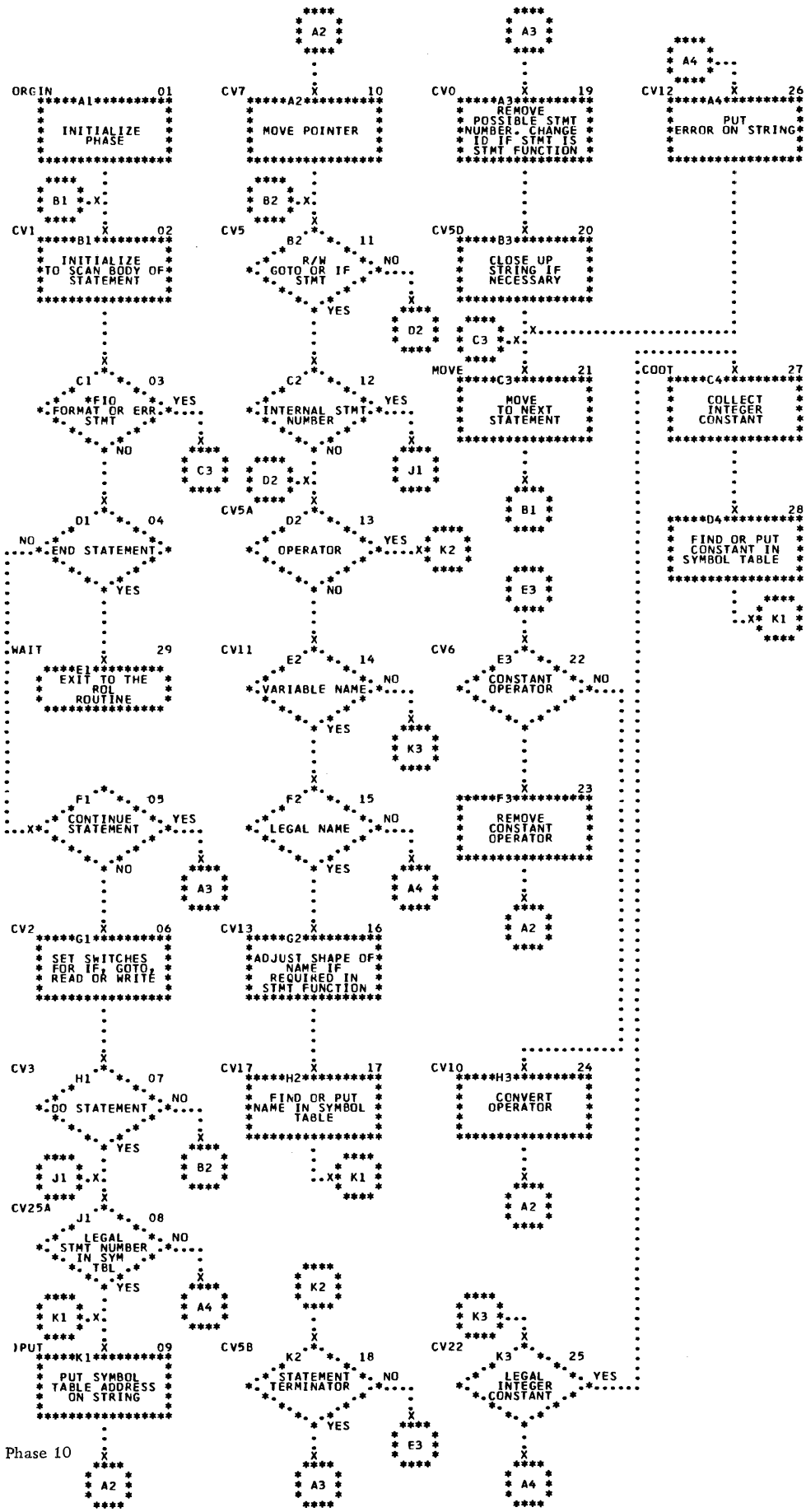


Chart DN. FORTRAN - Phase 10

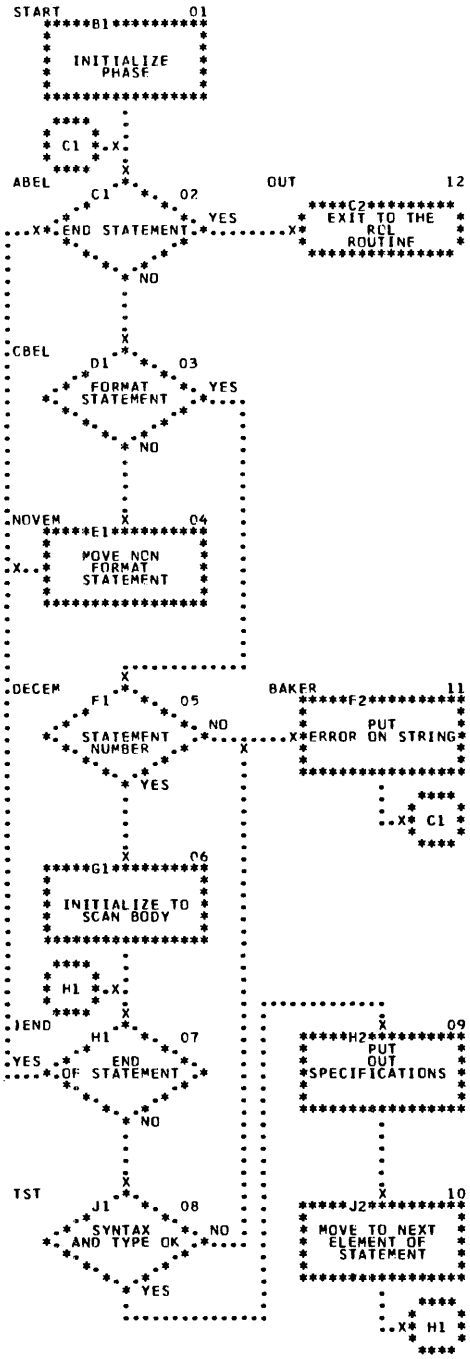


Chart DP. FORTRAN - Phase 11

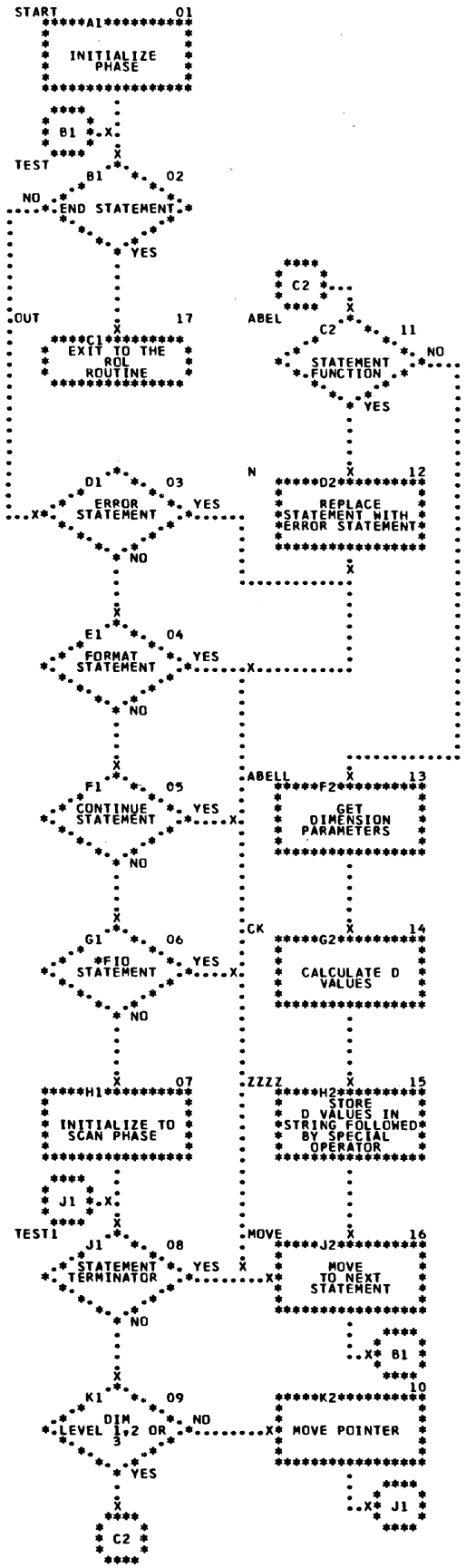


Chart DQ. FORTRAN - Phase 12

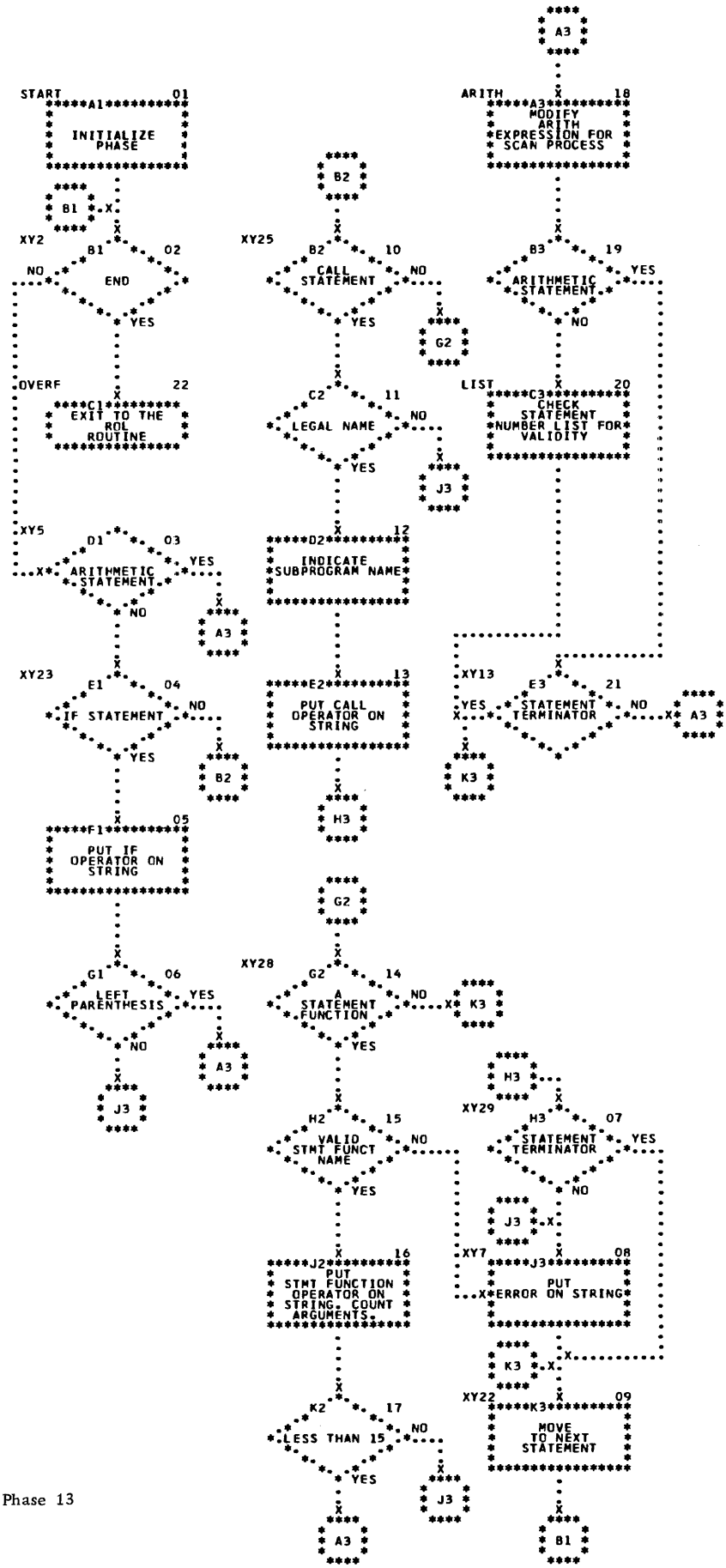


Chart DR. FORTRAN - Phase 13

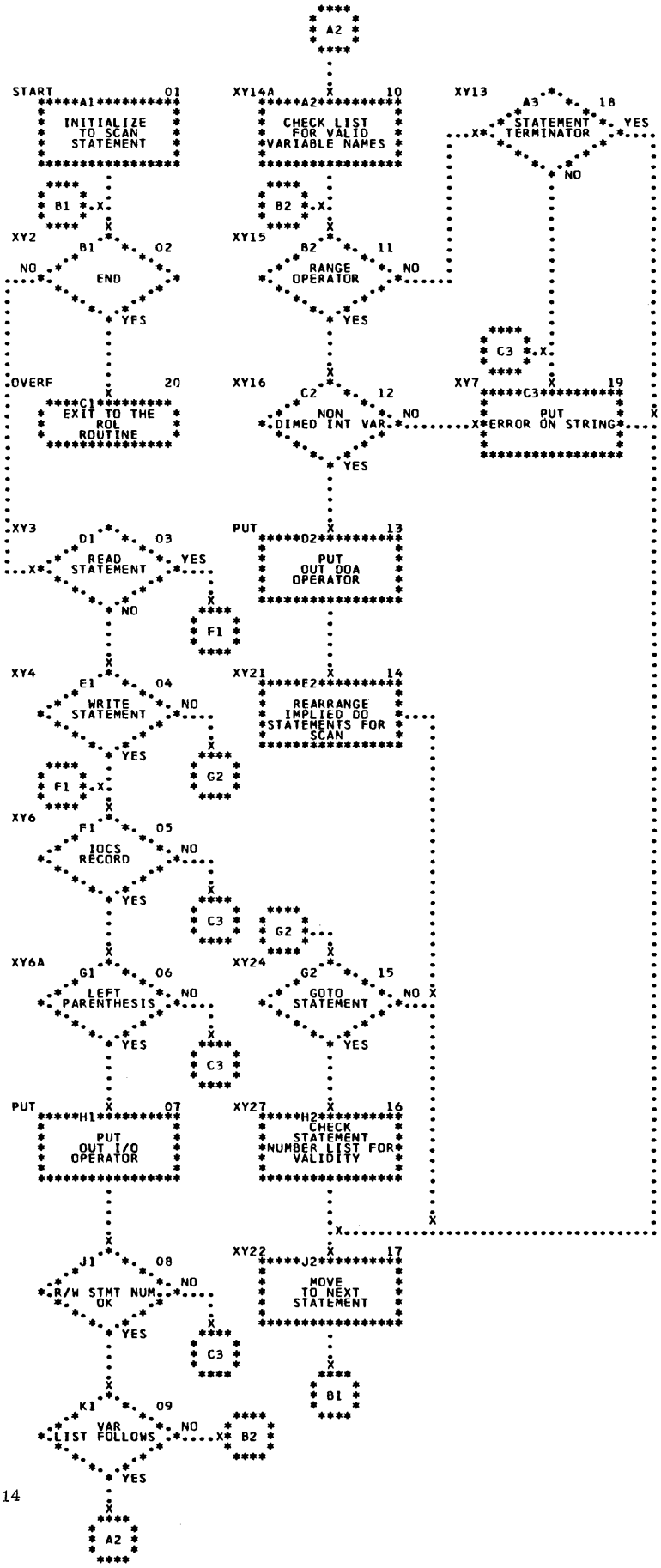


Chart DS. FORTRAN - Phase 14

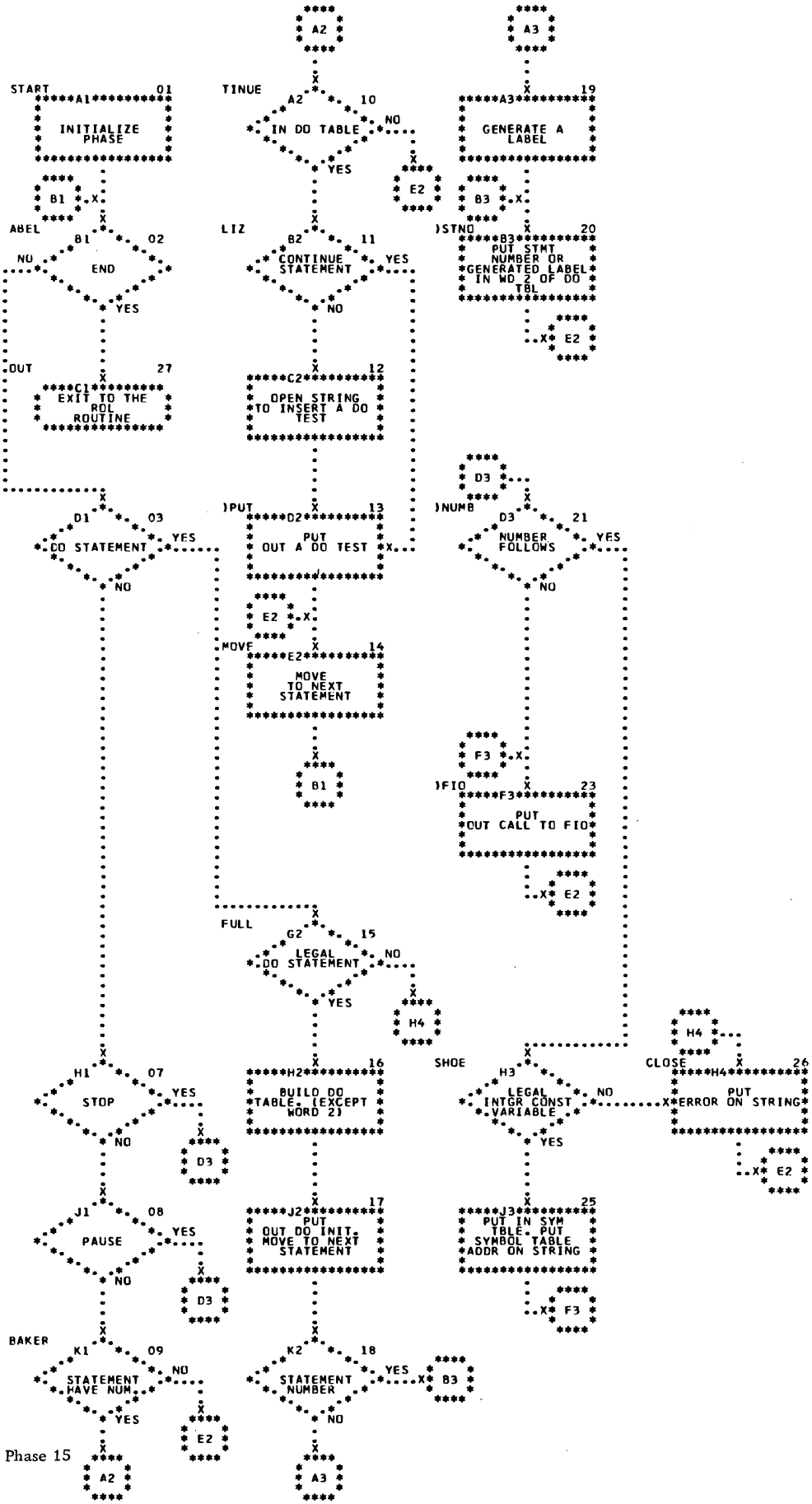


Chart DT. FORTRAN - Phase 15

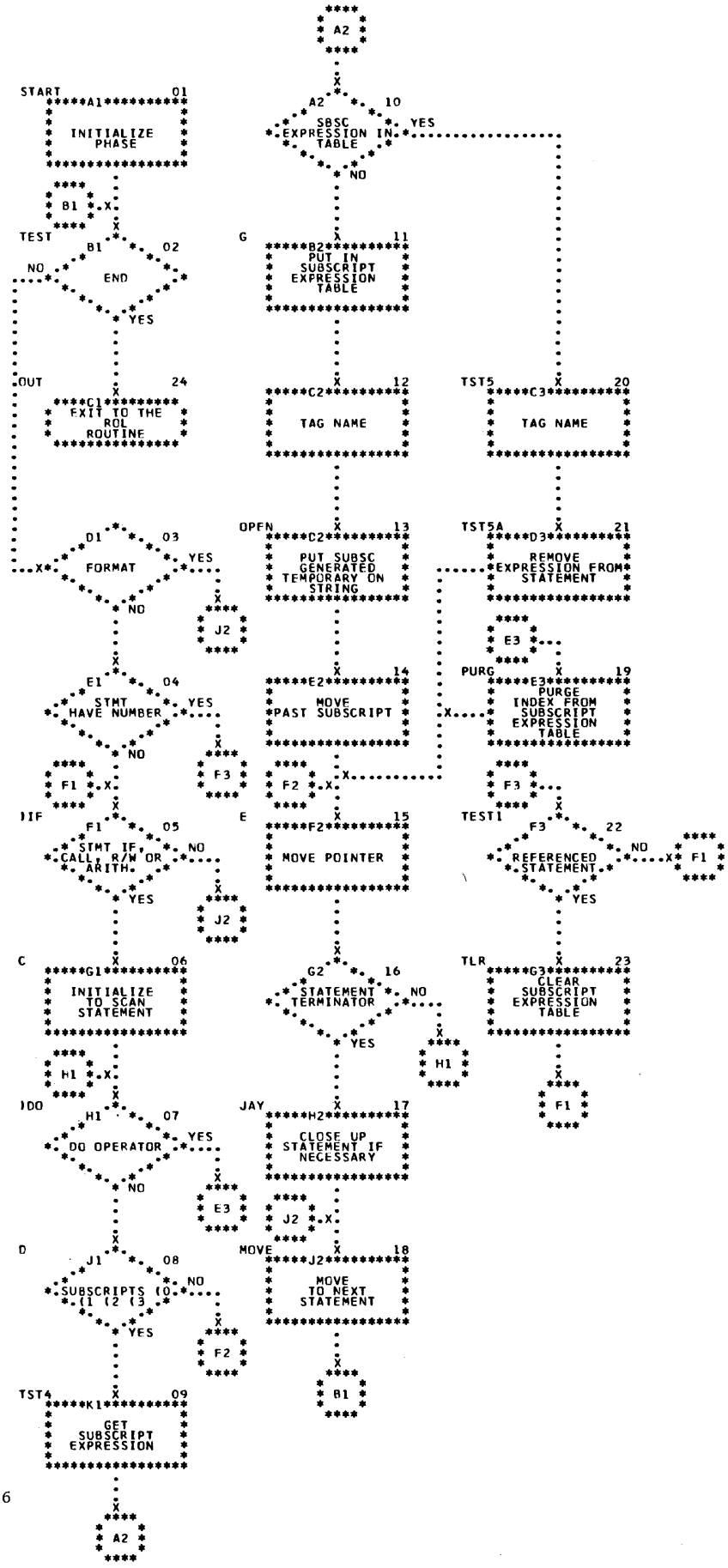


Chart DU. FORTRAN - Phase 16

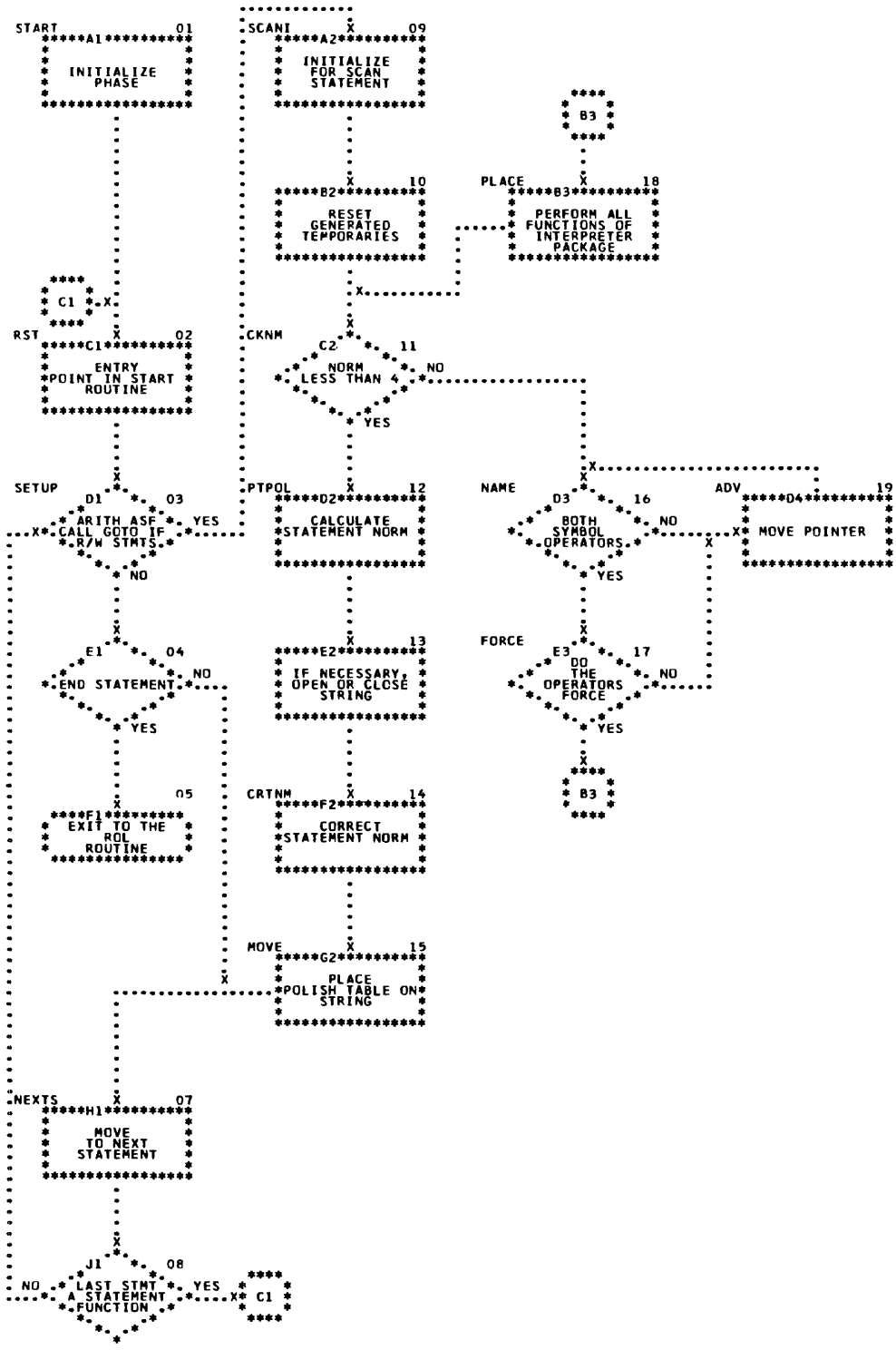


Chart DV. FORTRAN - Phase 17

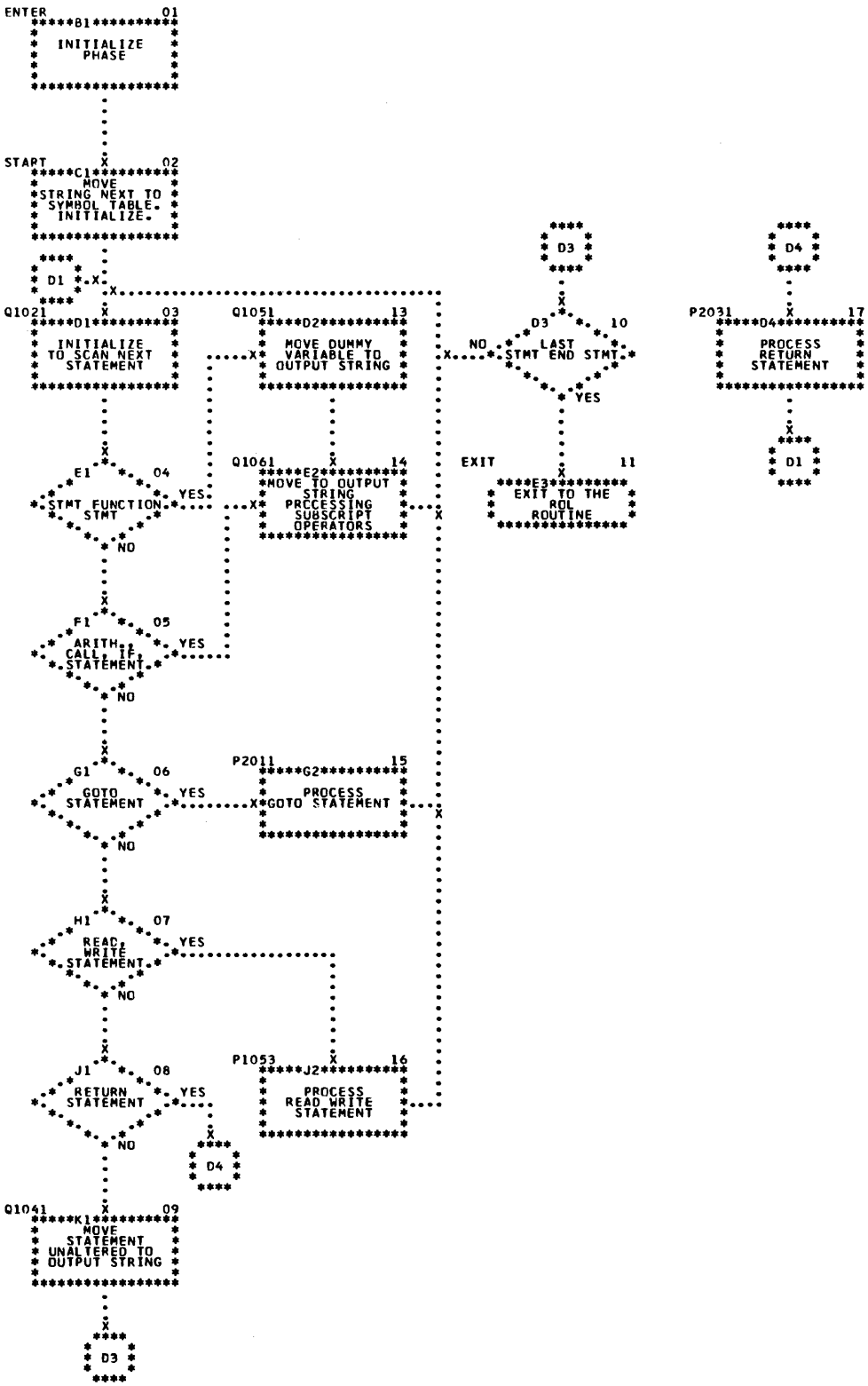


Chart DW. FORTRAN - Phase 18

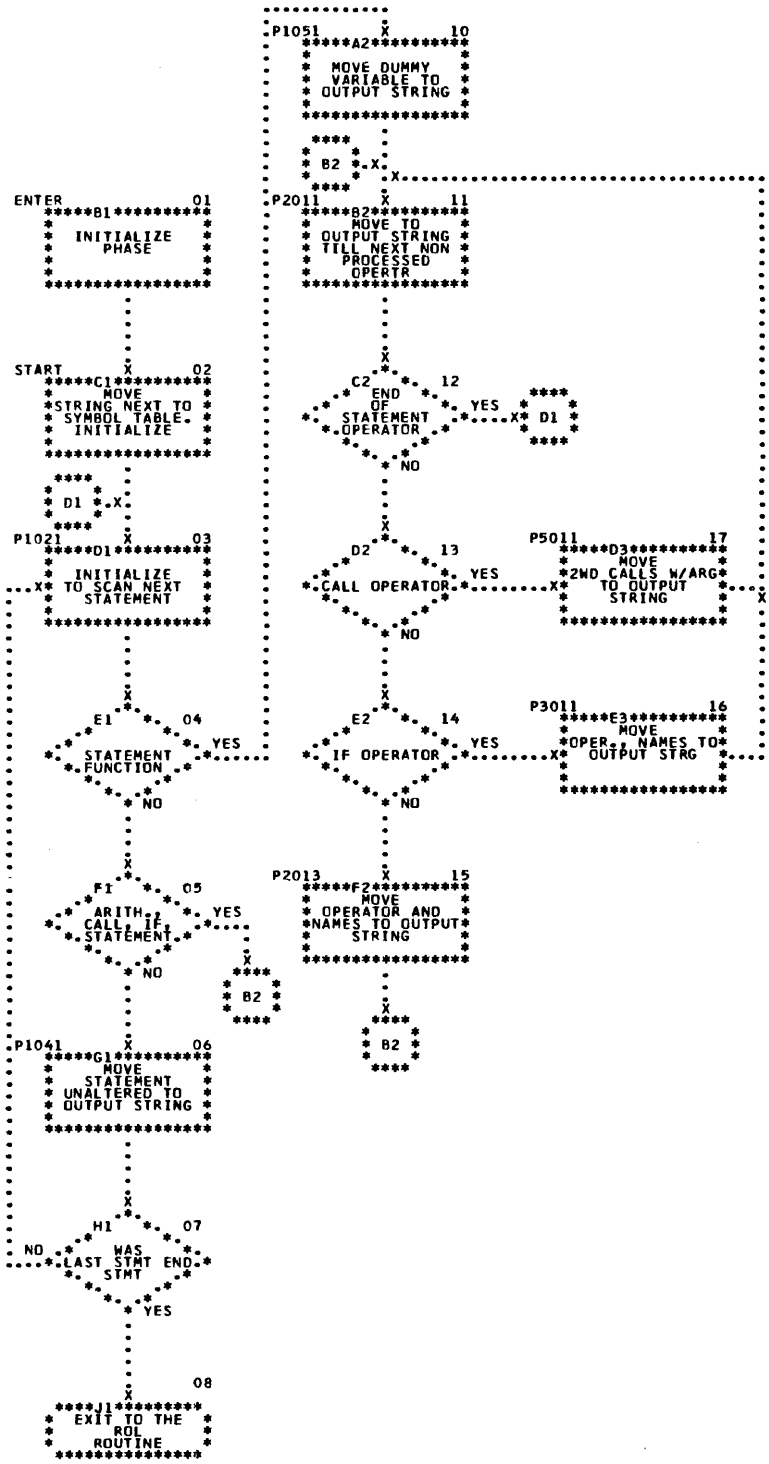


Chart DX. FORTRAN - Phase 19

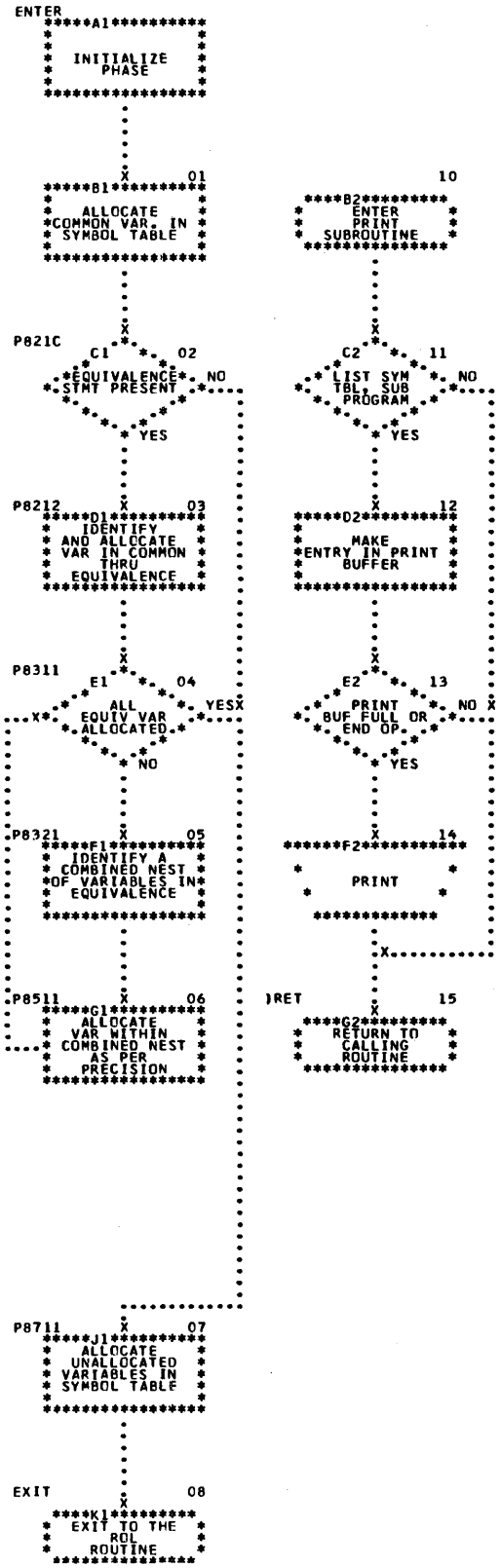


Chart DY. FORTRAN - Phase 20

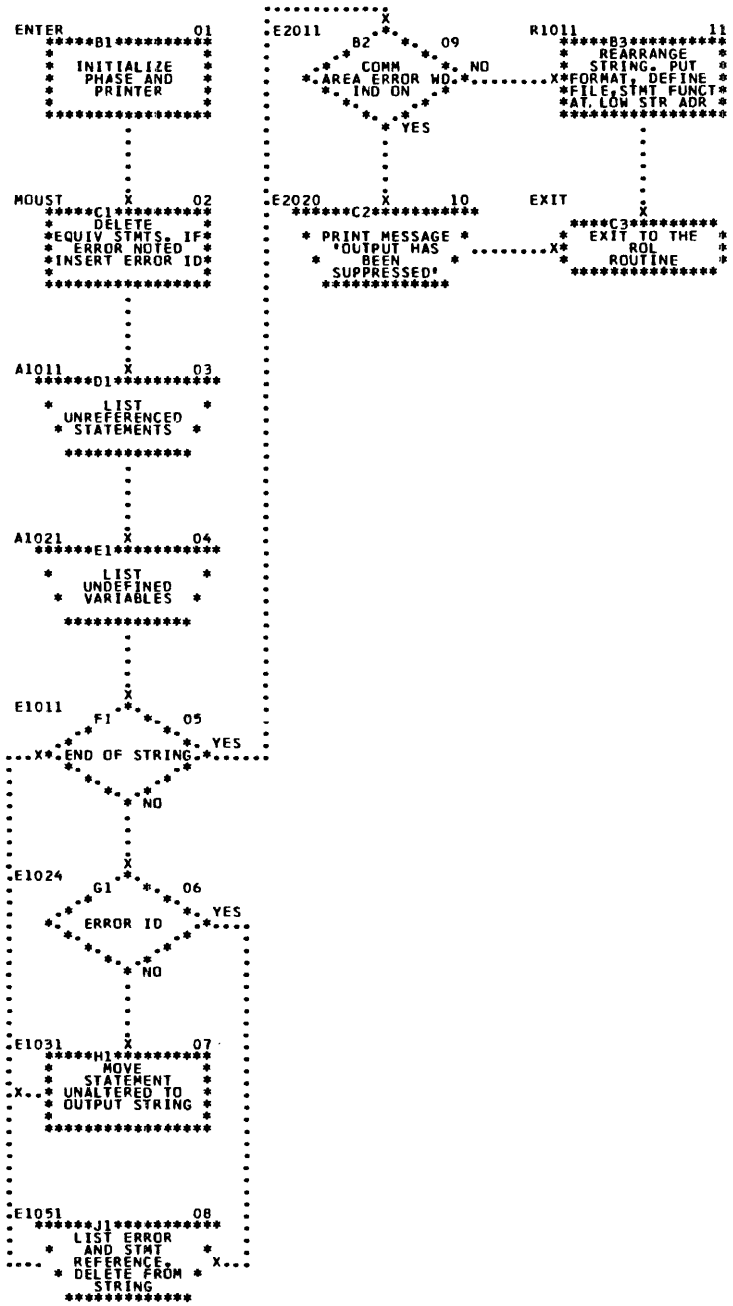


Chart DZ. FORTRAN - Phase 21

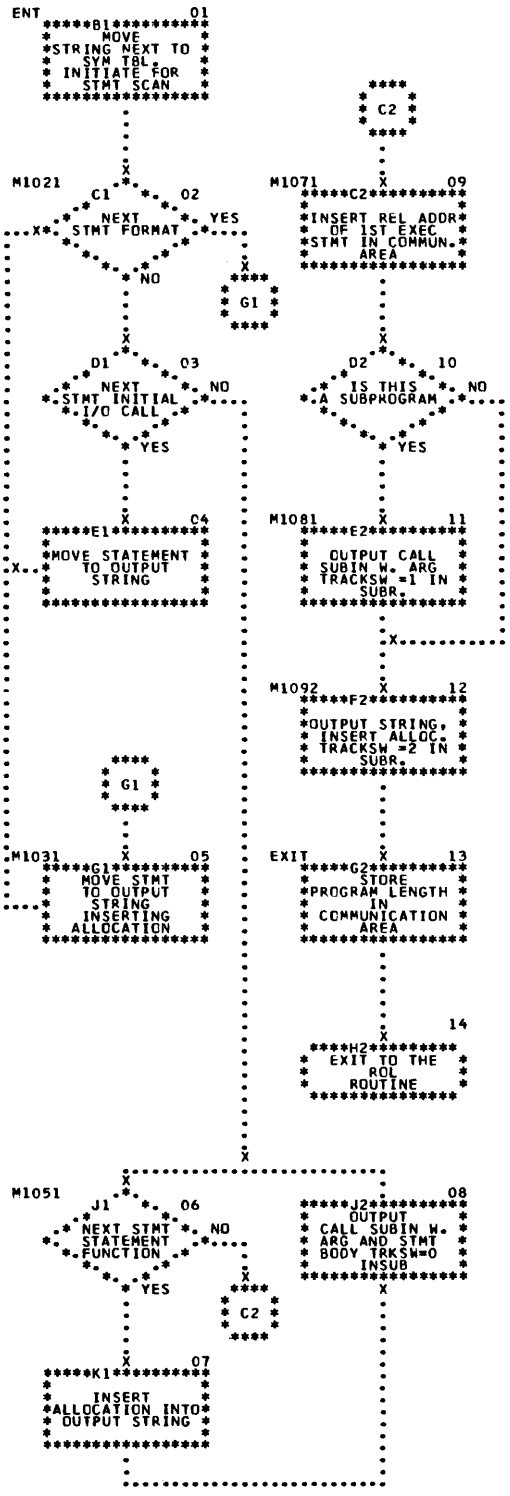


Chart EA. FORTRAN - Phase 22

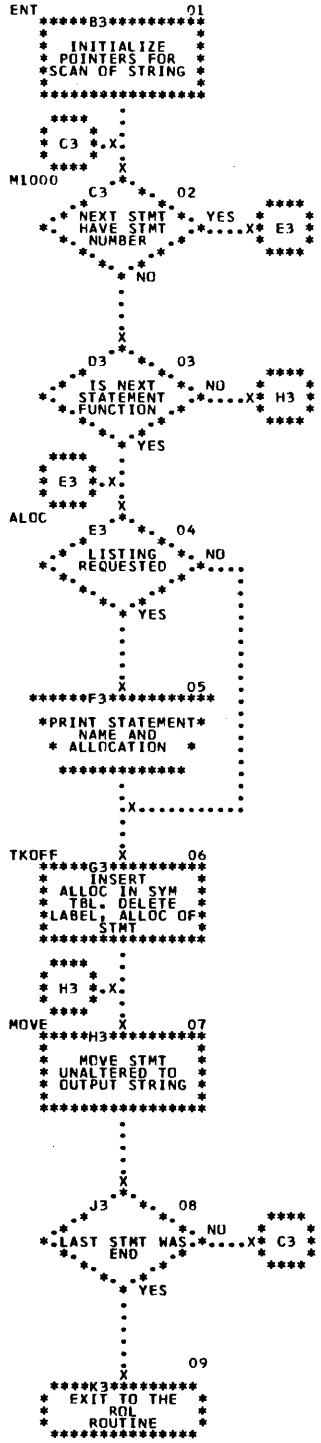


Chart EB. FORTRAN - Phase 23

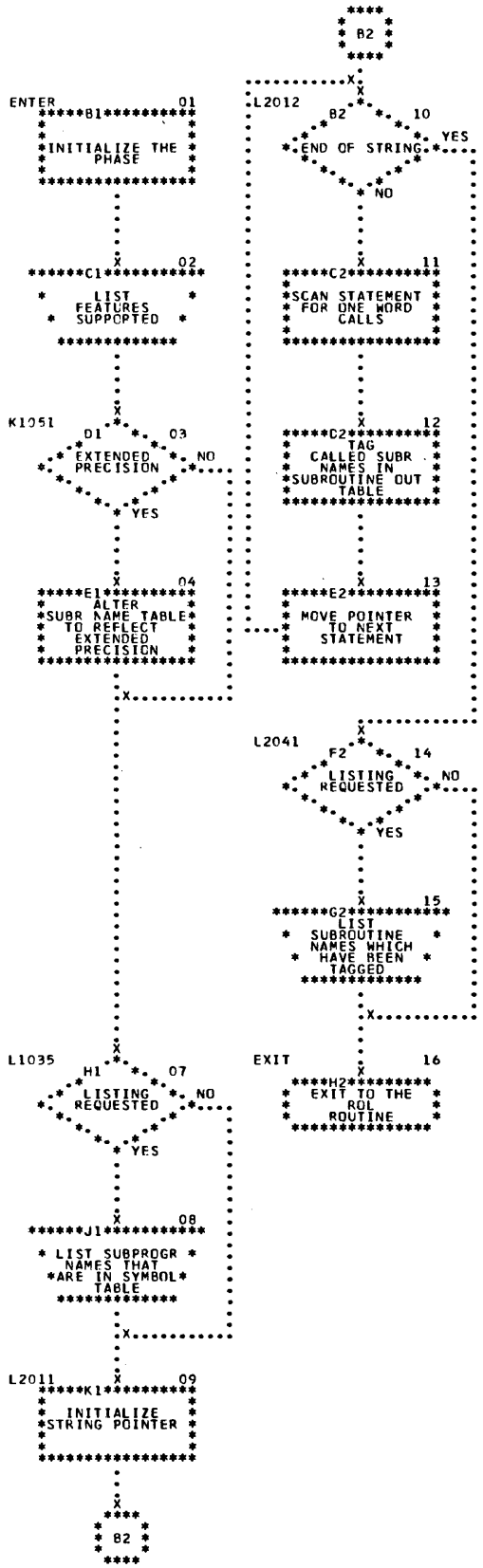


Chart EC. FORTRAN - Phase 24

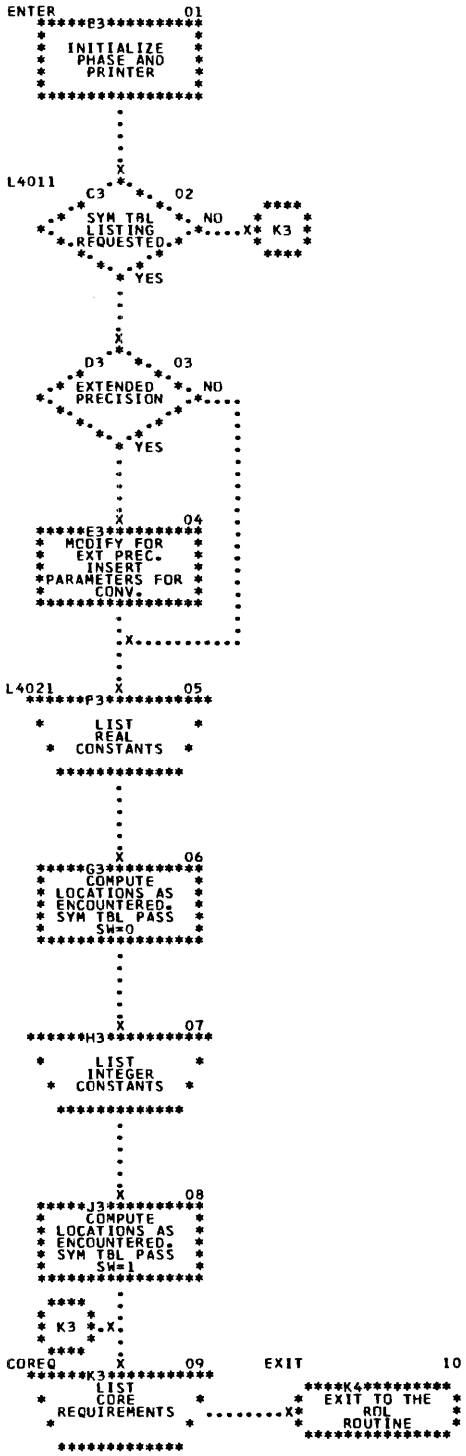


Chart ED. FORTRAN - Phase 25

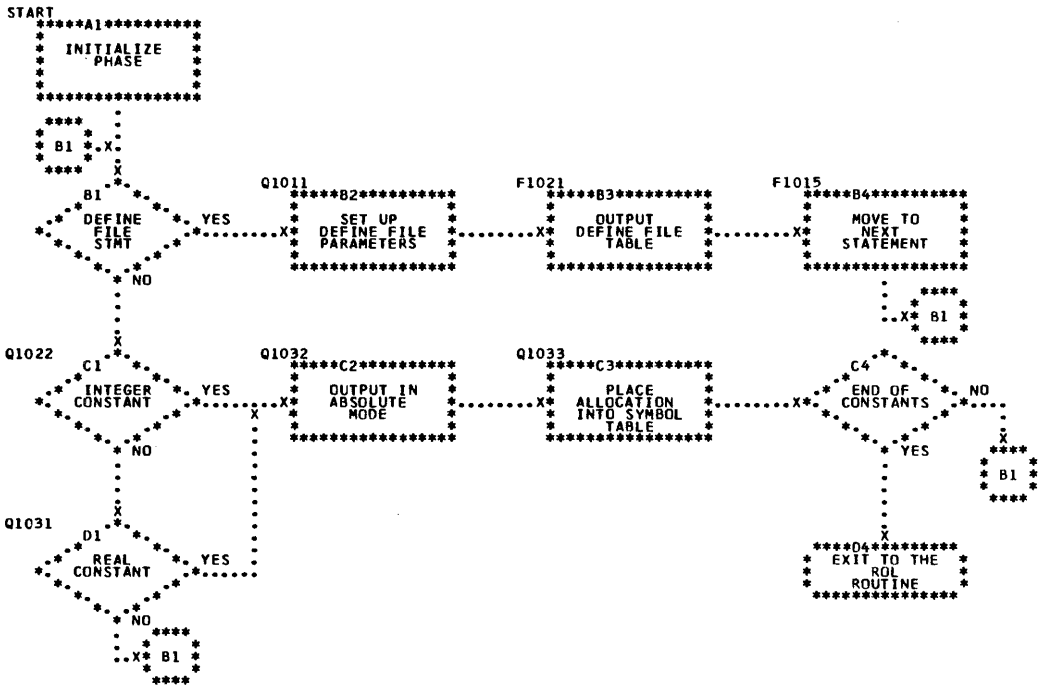


Chart EE. FORTRAN - Phase 26

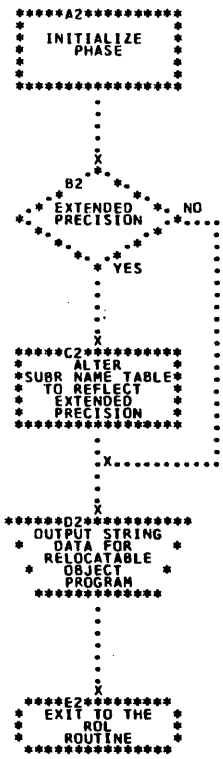


Chart EF. FORTRAN - Phase 27

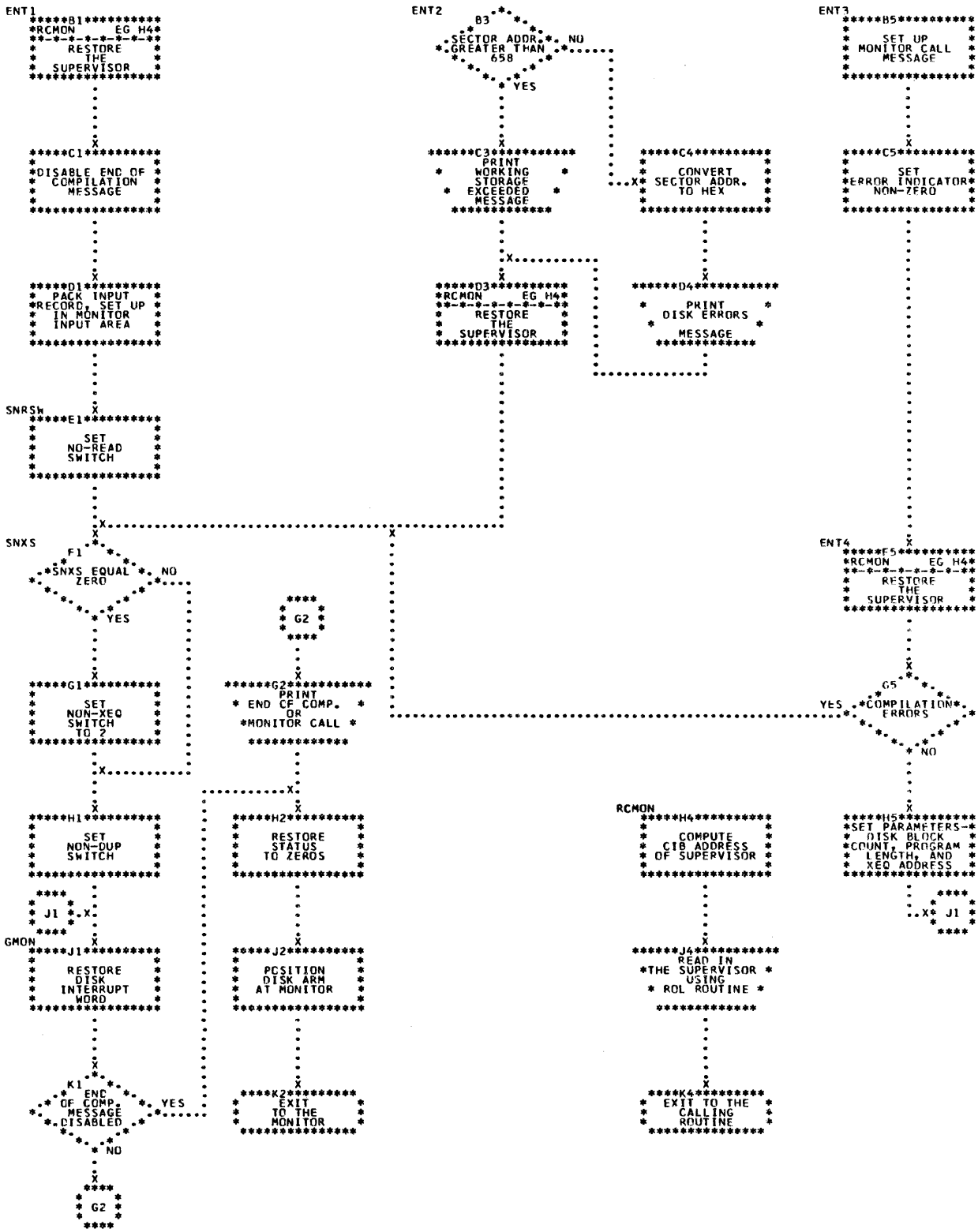


Chart EG. FORTRAN - Phase 28

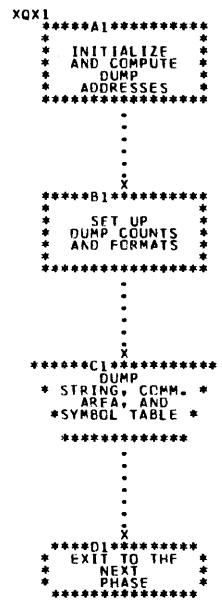


Chart EH. FORTRAN - Dump Phase

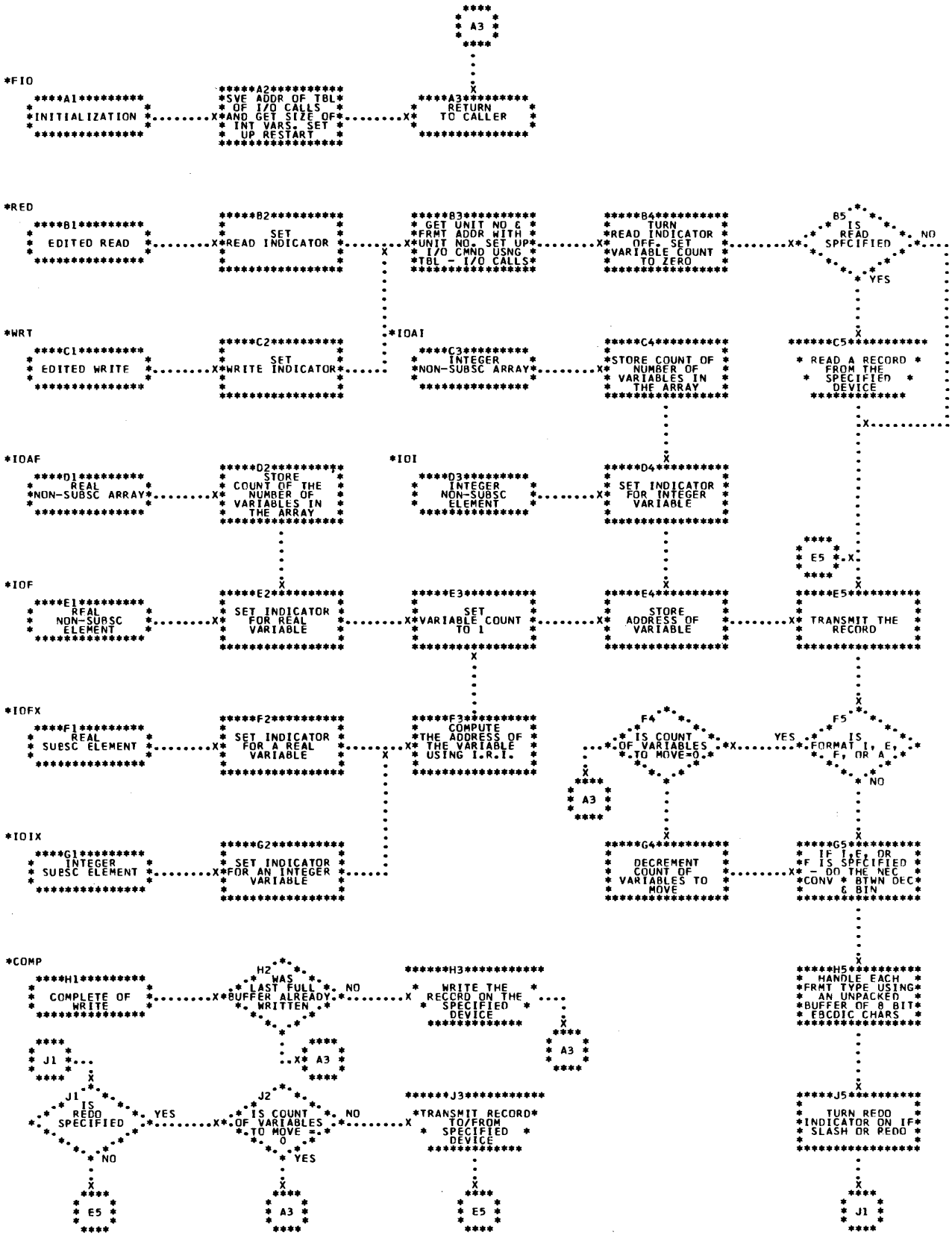


Chart FA. FORTRAN I/O

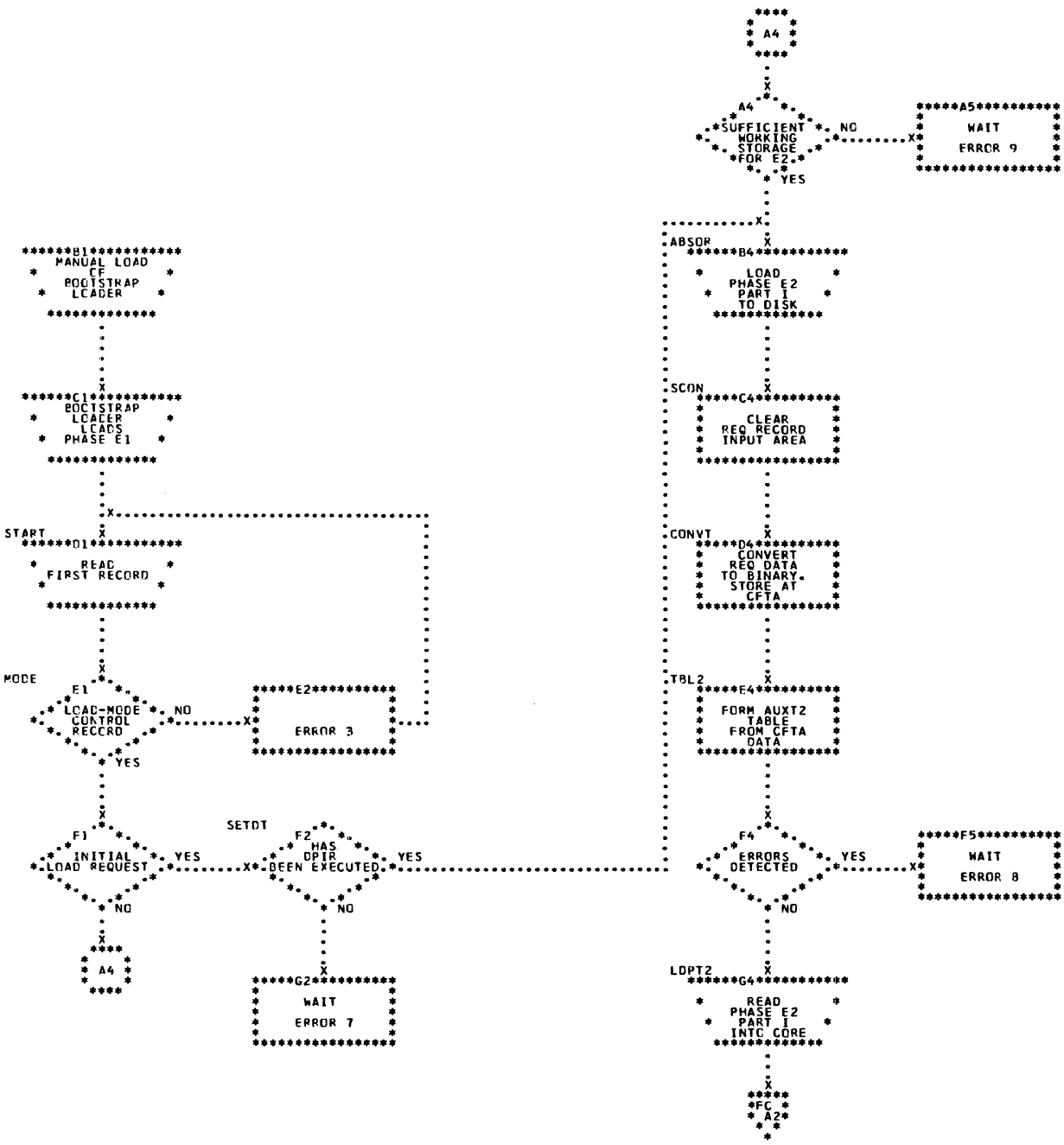


Chart FB. System Loader/Editor - Phase E1

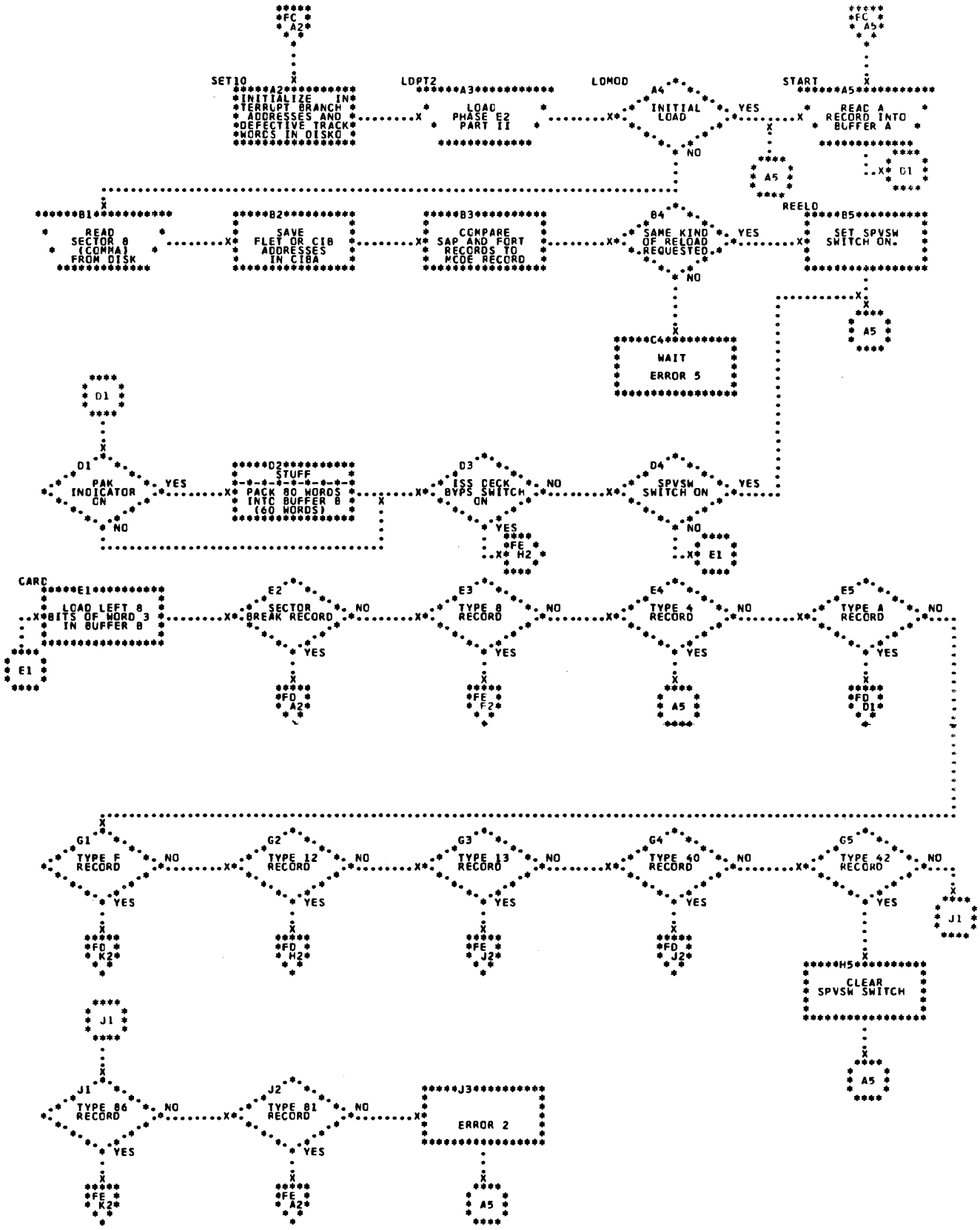


Chart FC. System Loader/Editor - Phase E2

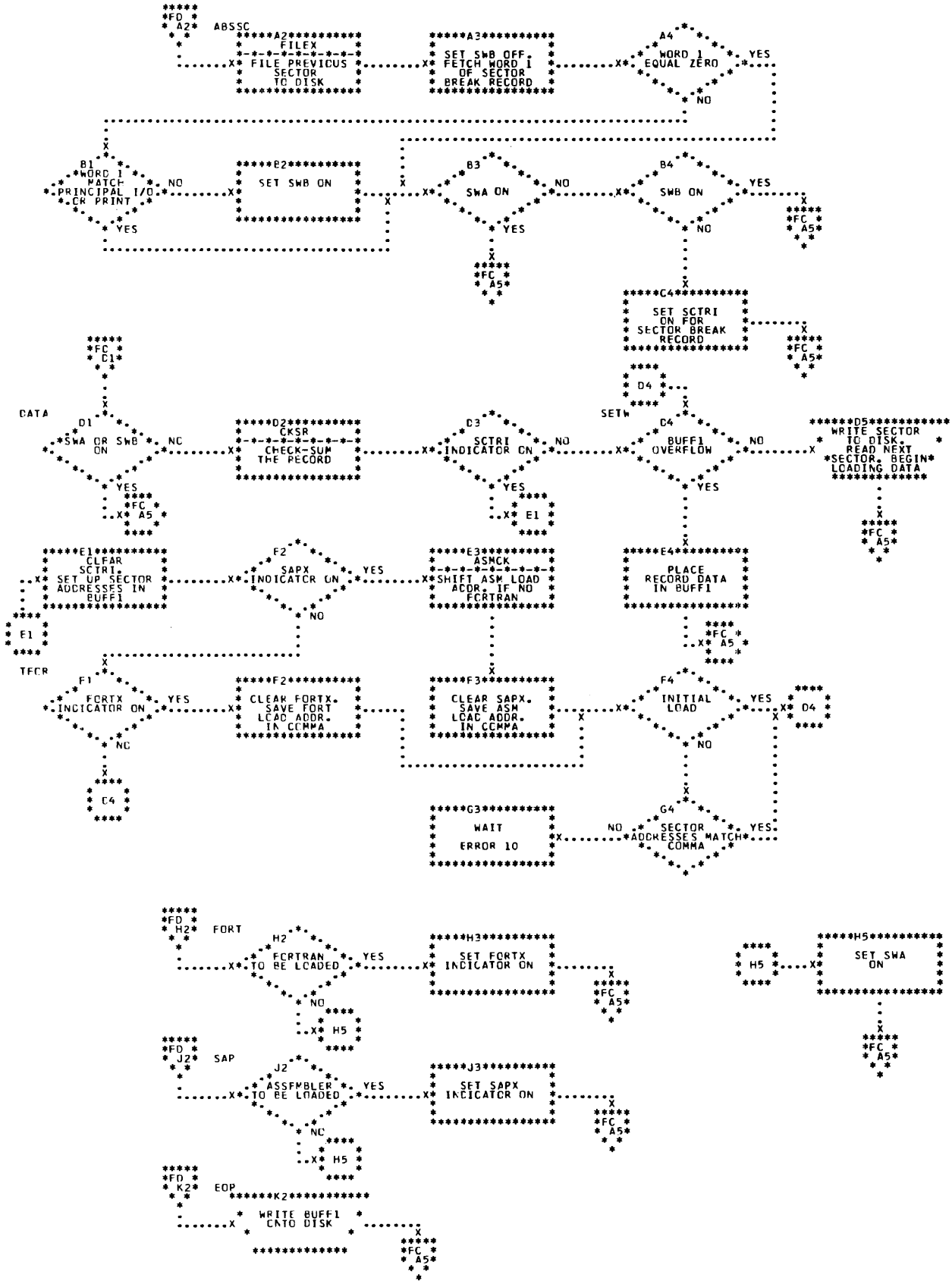


Chart FD. System Loader/Editor - Phase E2

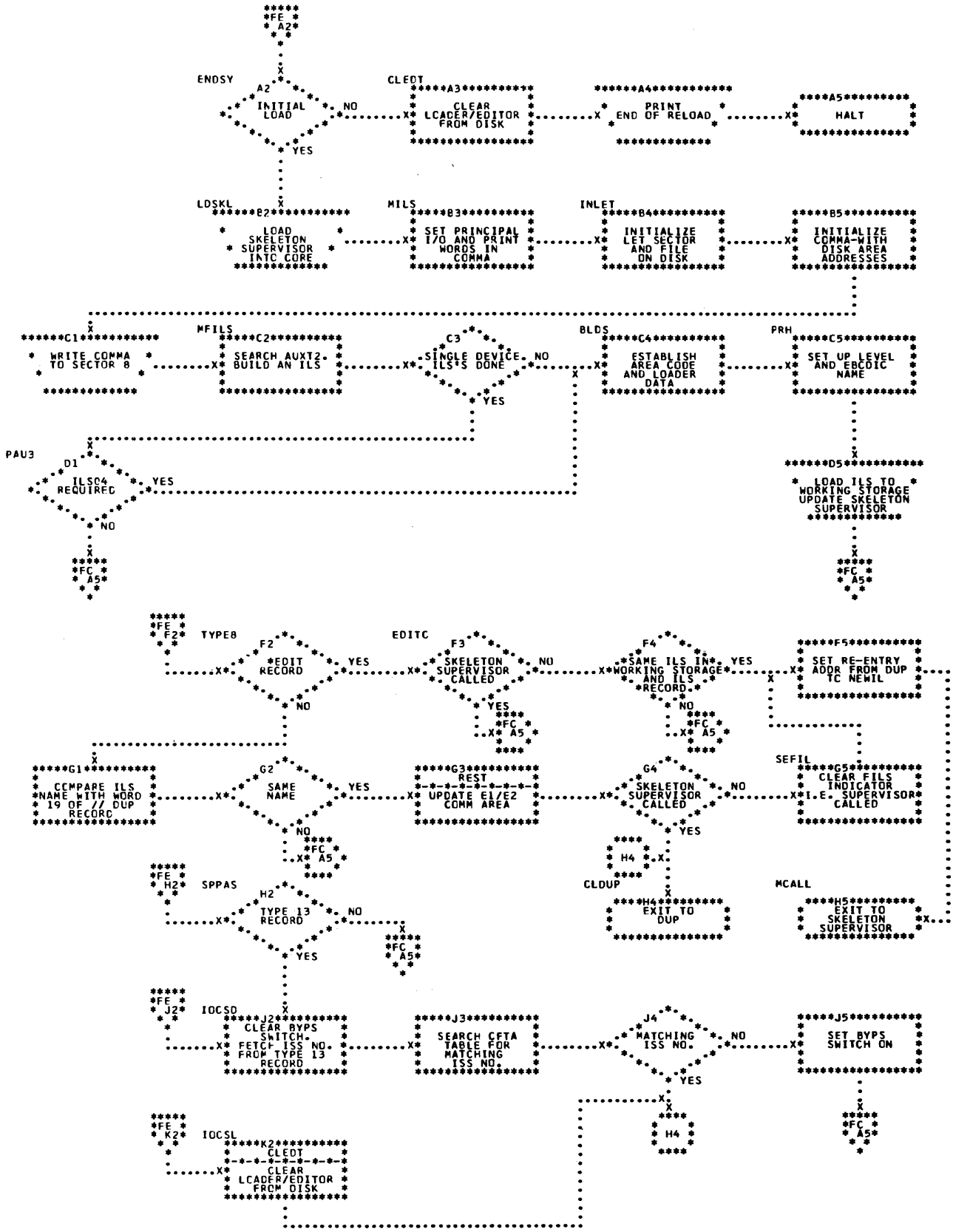


Chart FE. System Loader/Editor - Phase E2

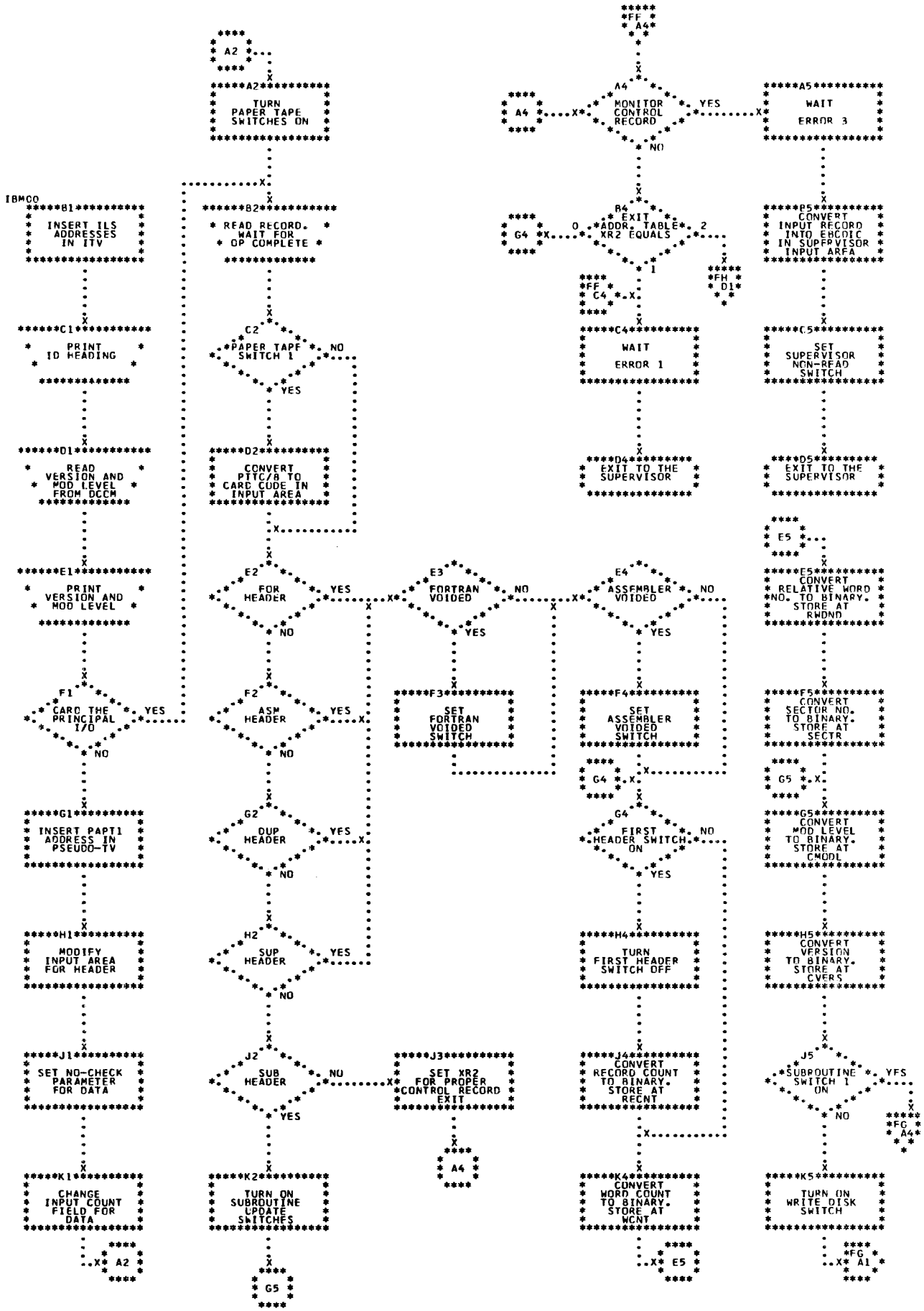


Chart FF. System Maintenance Program

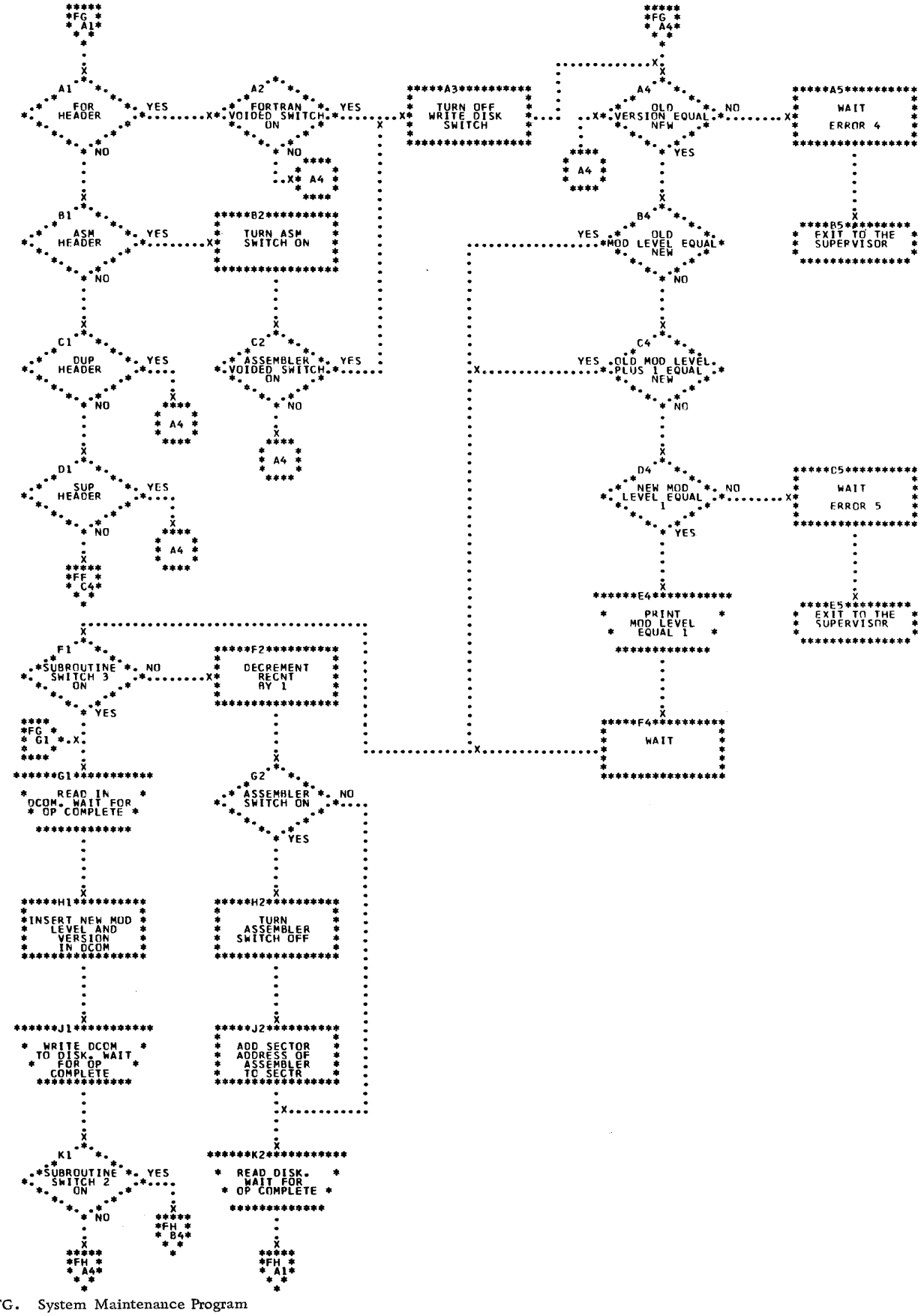


Chart FG. System Maintenance Program

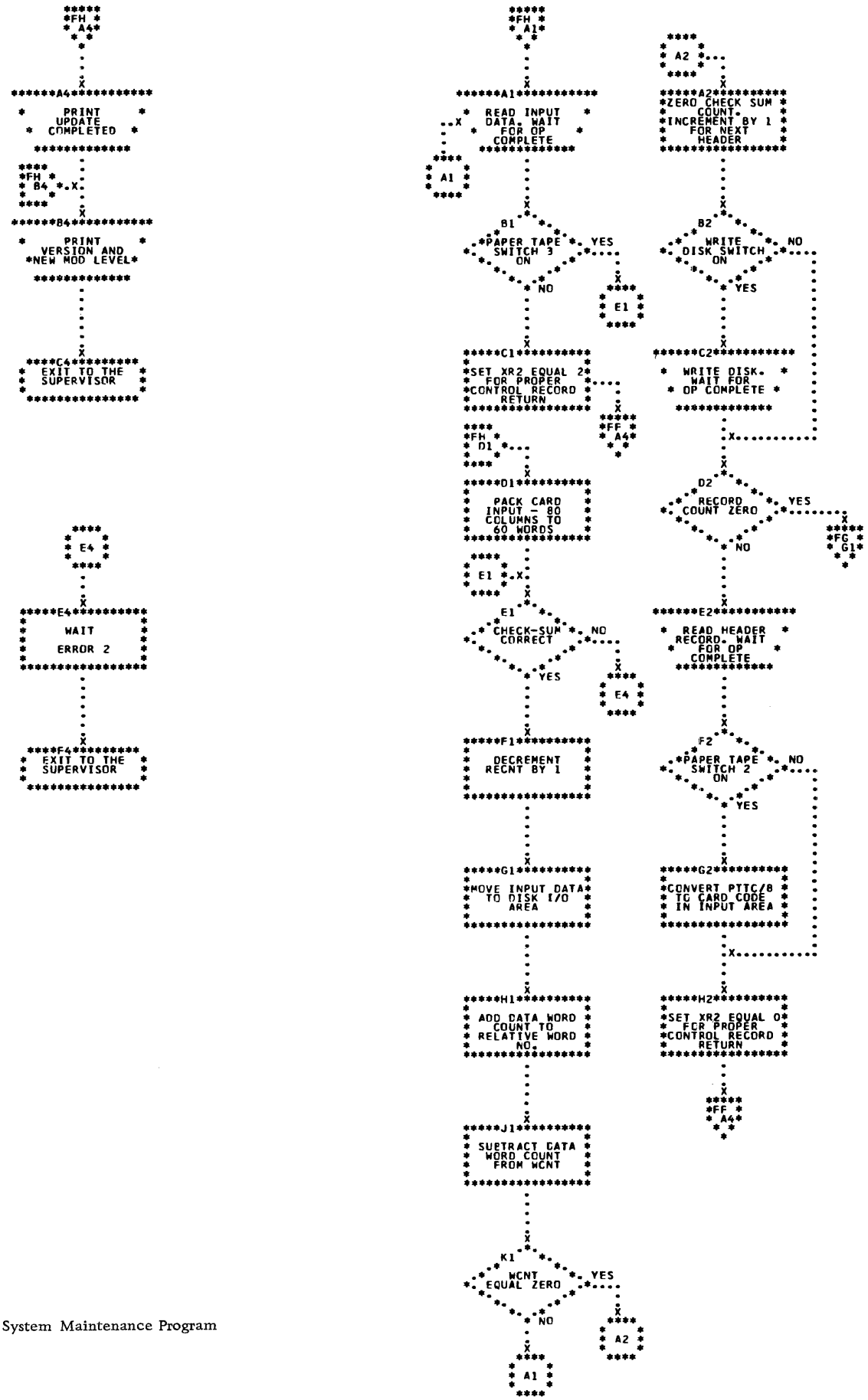


Chart FH. System Maintenance Program

This appendix shows by example the Assembler Language equivalent for the object coding generated by the FORTRAN Compiler. A typical cross-section of FORTRAN statements is shown.

Source Coding	Object Coding			With Trace*	
Arithmetic Statements - real, integer, and mixed modes					
I=J	LD	L	J	.	
	STO	L	I	LIBF	FIAR
				DC	I
A=B	LIBF		FLD	.	
	DC		B	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=I	LD	L	I	.	
	LIBF		FLOAT	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
				.	
I=A	LIBF		FLD	.	
	DC		A	.	
	LIBF		IFIX	.	
	STO	L	I	LIBF	FIAR
				DC	I
I=K-M	LD	L	K	.	
	S	L	M	.	
	STO	L	I	LIBF	FIAR
				DC	I
A=I-B	LD	L	I	.	
	LIBF		FLOAT	.	
	LIBF		FSUB	.	
	DC		B	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=B-I	LD	L	I	.	
	LIBF		FLOAT	.	
	LIBF		FSBR	.	
	DC		B	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	

*Period indicates that the generated coding is the same as in the Object Coding column.

Source Coding	Object Coding			With Trace	
A=B+I-J (or A=I+B-J)	LD	L	I	.	
	LIBF		FLOAT	.	
	LIBF		FADD	.	
	DC		B	.	
	LIBF		FSTO	.	
	DC		GT1	.	
	LD	L	J	.	
	LIBF		FLOAT	.	
	LIBF		FSBR	.	
	DC		GT1	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
I=J*K	LD	L	J	.	
	M	L	K	.	
	SLT		16	.	
	STO	L	I	LIBF	FIAR
				DC	I
A=B*C	LIBF		FLD	.	
	DC		B	.	
	LIBF		FMPY	.	
	DC		C	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=B*I	LD	L	I	.	
	LIBF		FLOAT	.	
	LIBF		FMPY	.	
	DC		B	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
I=J / K	LD	L	J	.	
	SRT		16	.	
	D	L	K	.	
	STO	L	I	LIBF	FIAR
				DC	I
A=B / C	LIBF		FLD	.	
	DC		B	.	
	LIBF		FDIV	.	
	DC		C	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
I=J / (K+M)	LD	L	K	.	
	A	L	M	.	
	STO	3	+126	.	
	LD	L	J	.	
	SRT		16	.	
	D	3	+126	.	
	STO	L	I	LIBF	FIAR
				DC	I

Source Coding	Object Coding			With Trace	
I=A / J	LD	L	J	.	
	LIBF		FLOAT	.	
	LIBF		FDVR	.	
	DC		A	.	
	LIBF		IFIX	.	
	STO	L	I	LIBF	FIAR
				DC	I
I=J**K	LD	L	J	.	
	LIBF		FIXI	.	
	DC		K	.	
	STO	L	I	LIBF	FIAR
				DC	I
A=B**I	LIBF		FLD	.	
	DC		B	.	
	LIBF		FAXI	.	
	DC		I	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=B**C	LIBF		FLD	.	
	DC		B	.	
	CALL		FAXB	.	
	DC		C	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=I**B	LD	L	I	.	
	LIBF		FLOAT	.	
	CALL		FAXB	.	
	DC		B	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=B**(I+J)	LD	L	I	.	
	A	L	J	.	
	STO	L	GT1	.	
	LIBF		FLD	.	
	DC		B	.	
	LIBF		FAXI	.	
	DC		GT1	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=B**(C+D)	LIBF		FLD	.	
	DC		C	.	
	LIBF		FADD	.	
	DC		D	.	
	LIBF		FSTO	.	
	DC		GT1	.	
	LIBF		FLD	.	
	DC		B	.	

Source Coding	Object Coding			With Trace	
	CALL		FAXB	.	
	DC		GT1	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
Subscripted Expressions					
A(I)=B(I,J)+C(I,J)	LIBF		SUBSC	.	
	DC		SGT1	.	
	DC		value D_4	.	
	DC		I	.	
(see Note 1)	DC		value D_1	.	
	LIBF		SUBSC	.	
	DC		SGT2	.	
	DC		value D_4	.	
	DC		J	.	
	DC		value D_2	.	
	DC		I	.	
(see Note 1)	DC		value D_1	.	
	LIBF		FLDX	.	
	DC		B	.	
	LIBF		FADDX	.	
	DC		C	.	
	LDX	I1	SGT1	.	
	LIBF		FSTOX	LIBF	FARIX
	DC		A	.	
M=L(I,J,K)	LIBF		SUBSC	.	
	DC		SGT1	.	
	DC		value D_4	.	
	DC		K	.	
	DC		value D_3	.	
	DC		J	.	
	DC		value D_2	.	
	DC		I	.	
(see Note 1)	DC		value D_1	.	
	LIBF		FLDX	.	
	DC		L	.	
	LIBF		FSTO	LIBF	FARI
	DC		M	.	
M(I)=M(I+1)+M(J)	LIBF		SUBSC	.	
	DC		SGT1	.	
	DC		value D_4	.	
	DC		I	.	
(see Note 1)	DC		value D_1	.	
	LIBF		SUBSC	.	
	DC		SGT2	.	
	DC		value D_4	.	
	DC		I	.	
(see Note 1)	DC		value D_1	.	
	LIBF		SUBSC	.	
	DC		SGT3	.	

Source Coding	Object Coding			With Trace	
	DC		value D_4	.	
	DC		J	.	
(see Note 1)	DC		value D_1	.	
	LDX	I1	SGT3	.	
	LD	L1	M	.	
	LDX	I1	SGT2	.	
	A	L1	M	.	
	LDX	I1	SGT1	.	
	STO	L1	M	LIBF	FIARX
				DC	M
M(1)=M(2)+M(3)	LDX	L1	value D_4	.	
	LD	L1	M	.	
	LDX	L1	value D_4	.	
	A	L1	M	.	
	LDX	L1	value D_4	.	
	STO	L1	M	LIBF	FIARX
				DC	M
M(1)=N(1)+M(1)	LDX	L1	value D_4	.	
	LD	L1	N	.	
	A	L1	M	.	
	STO	L1	M	LIBF	FIARX
				DC	M
Statement Function Statements					
A=JOE(B+C,D) +E	LIBF		FLD	.	
	DC		B	.	
	LIBF		FADD	.	
	DC		C	.	
	LIBF		FSTO	.	
	DC		GT1	.	
	CALL		JOE	.	
	DC		GT1	.	
	DC		D	.	
	LIBF		FLOAT	.	
	LIBF		FADD	.	
	DC		E	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
A=C(B,.5)+E	CALL		C	.	
	DC		B	.	
	DC		address of constant .5	.	
	LIBF		FADD	.	
	DC		E	.	
	LIBF		FSTO	LIBF	FARI
	DC		A	.	
CALL XY	CALL		XY	.	

Source Coding		Object Coding			With Trace
CALL YZ (A(2),A(I),B,C*D)		LIBF		SUBSC	.
		DC		SGT1	.
		DC		value D_4	.
		DC		I	.
(see Note 1)		DC		value D_1	.
		LIBF		FLD	.
		DC		C	.
		LIBF		FMPY	.
		DC		D	.
		LIBF		FSTO	.
		DC		GT1	.
		LDX	L1	value D_4	.
				(for A(2))	.
		MDX	L1	A	.
(see Note 2)		NOP			.
		STX	L1	ADR1	.
		LDX	I1	SGT1	.
		MDX	L1	A	.
(see Note 2)		NOP			.
		STX	L1	ADR2	.
		CALL		YZ	.
	ADR1	DC		0	.
	ADR2	DC		0	.
		DC		B	.
		DC		GT1	.
CALL YZ (A(I),B,C*D)		LIBF		SUBSC	.
		DC		SGT1	.
		DC		value D_4	.
		DC		I	.
(see Note 1)		DC		value D_1	.
		LIBF		FLD	.
		DC		C	.
		LIBF		FMPY	.
		DC		D	.
		LIBF		FSTO	.
		DC		GT1	.
		LDX	I1	SGT1	.
		MDX	L1	A	.
(see Note 2)		NOP			.
		STX	L1	ADR1	.
		CALL		YZ	.
	ADR1	DC		0	.
		DC		B	.
		DC		GT1	.

DO and CONTINUE Statements

DO 10 I=J,K		LD	L	J	.
:		STO	L	I	.
:	$\alpha \rightarrow$	:			.
10 CONTINUE		:			.
		:			.
		:			.

Source Coding		Object Coding			With Trace		
		MDX	L	I,1	.		
		LD	L	I	.		
		S	L	K	.		
		BSC	L	Adr. α , +	.		
DO 10 I =J,K,M		LD	L	J	.		
:		STO	L	I	.		
:	$\beta \rightarrow$	:			.		
10 CONTINUE		:			.		
		:			.		
		:			.		
		:			.		
		LD	L	I	.		
		A	L	M	.		
		STO	L	I	.		
		S	L	K	.		
		BSC	L	Adr. β , +	.		
GO TO Statement							
GO TO 111		BSC	L	111	.		
Computed GO TO Statement							
GO TO (111,112,113),I		LDX	I1	I	.		
	ADR1	BSC	I1	ADR1+1	LIBF	FGOTO	(see Note 3)
		DC		111	.		
		DC		112	.		
		DC		113	.		
IF Statements							
IF (I) 111,112,113		LD	L	I	.		
		BSC	L	111, +Z	LIBF	FIIF	(see Note 3)
		BSC	L	112, +-	BSC L	111,+Z	
		BSC	L	113, -Z	BSC L	112,+-	
					BSC L	113,-Z	
IF(A) 111,100,113		LIBF		FLD	.		
100 CONTINUE		DC		A	.		
		LD	3	+126	LIBF	FIF	(see Note 3)
		BSC	L	111,+Z	.		
		BSC	L	113,-Z	.		
IF(A+I) 100,111,100		LD	L	I	.		
100 CONTINUE		LIBF		FLOAT	.		
		LIBF		FADD	.		
		DC		A	.		
		LD	3	+126	LIBF	FIF	(see Note 3)
		BSC	L	111,+-			

Source Coding	Object Coding			With Trace			
IF(I) 111,111,112	LD	L	I				
100 CONTINUE	BSC	L	111,+	LIBF		FIIF	(see Note 3)
	BSC	L	112,-Z	BSC	L	111,+	
				BSC	L	112,-Z	

PAUSE Statement

PAUSE 11	LIBF		PAUSE
	DC		Adr. of constant 11

STOP Statement

STOP 21	LIBF		STOP
	DC		Adr. of constant 21

RETURN Statements

for FUNCTION-type return	LIBF		FLD
	DC		NAME
	BSC	I	subprogram linkword
or			
	LD	L	NAME
	BSC	I	subprogram linkword
for SUBROUTINE -type return	BSC	I	subprogram linkword

END Statement

The END statement produces no object code.

I/O Statements

The first I/O call is always 'LIBF *FIO' followed by the parameters and other I/O calls as indicated by the *IOCS control record.

LIBF	*FIO
DC	/00XY
DC	12
LIBF	WRTYZ
DC	0
LIBF	CARDZ
DC	0
LIBF	PRNTZ
DC	0
LIBF	PAPTZ
DC	0
DC	0
DC	0
LIBF	TYPEZ
DC	0

See the Subroutine Library section for a detailed discussion of FORTRAN I/O routines.

Source Coding	Object Coding		
READ Statement			
READ (N,101) A,I	LIBF		*RED
	DC		N
	DC		101
	LIBF		*IOF
	DC		A
	LIBF		*IOI
	DC		I
READ (N,101) X (X dimensioned)	LIBF		*RED
	DC		N
	DC		101
	LIBF		*IOAF
	DC		X
	DC		(number of items in array)
Read Statement with Implied DO			
READ (N,101)(X(I), I=1,5)	LIBF		*RED
	DC		N
	DC		101
	LD	L	Adr. of constant 1
	STO	L	I
α)→	LIBF		SUBSG
	DC		SGT1
	DC		value D_4
	DC		I
	DC		value D_1
	LDX	I1	SGT1
(see Note 1)	LIBF		*IOFX
	DC		X
	MDX	L	I,1
	LD	L	I
	S	L	Adr. of constant 5
	BSC	L	Adr. α , +Z
Write Statements			

Write statements closely parallel read statements, except for the 'LIBF *COMP' which terminates all WRITE calls.

WRITE (N,101) A,I	LIBF		*WRITE
	DC		N
	DC		101
	LIBF		*IOF
	DC		A
	LIBF		*IOI
	DC		I
	LIBF		*COMP

NOTES 1: Tagged to indicate the end of the subscript argument list.
2: Included to balance skip at execution of previous instruction.
3: Transfer Trace.

APPENDIX B. DIAGNOSTIC AIDS

The following procedures, in conjunction with Appendix C, are intended to assist the user in defining and analyzing System problems. These procedures are a guide to determining which System element is in control at any given time.

To determine which Monitor Pogram is in control, examine the control records printed. The last Monitor control record printed indicates the controlling Monitor Program.

To determine which phase of the Monitor Program is in control, perform the following:

Supervisor

1. Display the location DSA+2 (00D0). This location contains the sector address of the first sector of the Supervisor phase in control.

Loader

1. Compare the first few words following the phase origin with the phase listing to determine if the phase has been loaded. (See Figure 4, Section 2.)
2. Once it is determined the phase is loaded, locate the phase entry point. If the entry point is zero, the phase has not been entered. If the entry point is non-zero, the phase has been entered. However, because of the overlays used in the Loader, the phase, even though it has been entered, is not necessarily in control.

DUP

1. Display the contents of locations 03B6 and 03B7.
2. Compare these two words to the table below. Each pair of words forms the IOAR header for a DUP function. The header consists of the word count (word 1) and the sector address (word 2) of the DUP function.
3. If the words are identical, display the next pair (03B8 and 03B9) and compare them to the table.
4. If the words are not identical, the DUP function whose IOAR header precedes the unmatched words is the function presently in core. In other words, each function is loaded starting at the first word following its IOAR header, except DCTL which starts at the first word following the IOAR header table.

IOAR Header Table

Core Locations	Word Count	Sector Address	DUP Function
03B6-7	047A	003C	DCTL
03B8-9	0476	0040	STORE
03BA-B	0460	0048	DUMP
03BC-D	0280	004C	DUMPLET
03BE-F	0500	0050	DELETE
03C0-1	03C0	0054	DEFINE
03C2-3	0BCC	0632	EDIT
03C4-5	0140	0057	DWADR
03C6-7	0280	0044	FILEQ
03C8-9	0140	0046	STOREMOD

Assembler

1. Display the contents of the location OVRLY+1 (05B1).
2. Subtract the sector address of Phase 0 (00E8 with FORTRAN, 0080 without FORTRAN).
3. The remainder is the displacement, in sectors, from Phase 0. The phase within which this displacement falls is presently in core.
4. Phase 10 resides in core in place of the Symbol Table add routine, STADD, during Phase 2. If the assembly is in one pass mode, Phase 11 replaces the subroutine INT1 in Phase 9.
5. The input phase (CARD0 or PAPT1) resides in core in Pass 1, but is overlaid by output in Disk System format in Pass 2 of a one pass mode assembly.
6. The listing routine (1132 Print or WRTY0) resides in core if an *LIST control record is part of the source input; otherwise, it is overlaid by the Symbol Table.

FORTRAN

1. Display the contents of location ROC3 (7FID).
2. Add the value /0140 (320_{10}) to the value found in ROL3.
3. Display the contents of the address computed above and the next higher addressed word. These two words comprise a disk IOCC. The second word of the disk IOCC (the computed address + 1) is the address, in hexadecimal, of the first sector of the last phase read; hence the phase presently in core.

SYSTEM LOADER

1. The card deck or paper tape strip read or to be read indicates which phase is in control.
2. During Phase E1, location A (0658) is the start of the input buffer; location 021B contains the sector address of the last sector written; location 0215 contains the sector address of the last sector read.
3. During Phase E2, location 053F contains the sector address of the last sector read.

APPENDIX C. DISK MAP

The following table is a breakdown of the 1130 Monitor System. For purposes of completeness, it is assumed that both the FORTRAN Compiler and the Assembler program are present, a Fixed Area of five cylinders is assigned with the resultant FLET, and a User Area of 30 cylinders is assigned.

SYSTEM ELEMENT	SECTOR ADDRESS DEC	SECTOR ADDRESS HEX	CORE ORIGIN (HEX)
Disk Pack ID	0	0	----
Cold Start routine	1	1	0802
Supervisor			
Phase E	2	2	055C
Phase B	4	4	055C
Phase D	6	6	055C
Skeleton Sup. and DCOM	8	8	0090
Console Printer and DISK0 routines	9	9	00F4
PAPT1 or CARD0	11	B	00F4
Phase C	12	C	055C
Presupervisor and Multi-Sector routines	13	D	04C0
Conversion routines, Phase A, and 1132 Printer routine	14	E	055C
Reserved	20	14	----
Loader			
Phase 0	24	18	0578
Phase 7	25	19	0BBC
Phase 8	26	1A	01C2 with DISKZ, 0260 with DISK0, 0370 with DISK1, and 0438 with DISKN.
DISK1	27	1B	00F4
DISKN	29	1D	00F4
DISKZ	32	20	00F4
Save Monitor routine	33	21	0090
Phase 1	34	22	0630
Phase 4	38	26	0BBC
Phase 5	39	27	0BBC
Phase 2	40	28	0944
Phase 3	44	2C	0BBC
Phase 6	47	2F	0BBC
Map1	48	30	026A
Map2	49	31	026A
Message 1	50	32	026A
Message 2	51	33	026A
Message 3	52	34	026A
1132 Printer/Console Printer	53	35	026A
Reserved	54	36	----
DUP			
DUPCO	56	38	0272
DCTL	60	3C	03CA
STORE	64	40	03BA
FILEQ	69	45	03C8
STOREMOD	71	47	03CA
SRLET	72	48	06A0
DUMP	73	49	03BC
DUMPLET	77	4D	03BE
Reserved	79	4F	----
ERM	80	50	045C
DELETE	81	51	03C0
DEFINE	85	55	03C2
DWADR	88	58	03C6
Principal I/O Device routine	89	59	082E
Principal Print Device routine	92	5C	0C4E
Principal Error Message Device routine	94	5E	0CFE
PTX	97	61	0828
Reserved	100	64	----
DUPCO Temp. Storage	109	6D	----
Reserved	110	6E	----

SYSTEM ELEMENT	SECTOR ADDRESS DEC	SECTOR ADDRESS HEX	CORE ORIGIN (HEX)
FORTRAN			
Phase 1	128	80	7EBC
Dump	129	81	7984
Reserved	131	83	----
Phase 28	142	8E	7590
Phase 27	144	90	7984
Phase 26	147	93	7984
Phase 25	150	96	7984
Phase 24	154	9A	7984
Phase 23	158	9E	7984
Phase 22	161	A1	7984
Phase 21	164	A4	7984
Phase 20	168	A8	7984
Phase 19	172	AC	7984
Phase 18	176	B0	7984
Phase 17	179	B3	7984
Phase 16	183	B7	7984
Phase 15	187	BB	7984
Phase 14	191	BF	7984
Phase 13	194	C2	7984
Phase 12	197	C5	7984
Phase 11	200	C8	7984
Phase 10	203	CB	7984
Phase 9	206	CE	7984
Phase 8	207	D0	7984
Phase 7	211	D3	7984
Phase 6	215	D7	7984
Phase 5	218	DA	7984
Phase 4	220	DC	7984
Phase 3	224	E0	7590
Phase 2	231	E7	7984
Assembler			
Phase 0	232	E8	0542
Phase 9	233	E9	0774
1132 Print/WRTY0	239	EF	0DDB
CARD0/PAPT1	240	F0	025E
Phase 1	242	F2	05B2
Phase 12	244	F4	05B2
Phase 10	246	F6	0C81
Phase 11	247	F7	0D0B
Phase 5	248	F8	05B2
Phase 6	250	FA	05B2
Phase 7	252	FC	05B2
Phase 8	254	FE	05B2
Phase 2	256	100	05B2
Phase 3	258	102	05B2
Phase 4	260	104	05B2
VIP/TYPE ER	262	106	03F6
Phase 1A	263	107	05B2
Phase 12A	264	108	05B2
System Symbol Table	265	109	0ED3 in a 4096-word machine; 1ED3 in an 8192-word machine
Reserved	266	10A	----
FLET	272	110	----
Fixed Area	280	118	----
CIB	320	140	----
LET	336	150	----
User Area	334	158	----
Working Storage	584	248	----
System Loader/Editor			
Bootstrap Loader	---	---	0000
Phase E1	---	---	0028
Phase E2 (Part I) *	1586	632	03C2
Phase E2 (Part II)*	1584	630	0100

*Phase E2 is loaded onto the disk by Phase E1 for the duration of the System Load only.

Absolute program: A program which, although in Disk System format, has been written in such a way that it can be executed from only one core location.

Assembler core load: A core load which was built from a mainline written in Assembly Language.

CALL routine: A routine which must be referenced with a CALL statement. The type codes for routines in this category are 4 and 6.

CALL TV: The transfer vector through which CALL routines are entered at execution time. See the section on the Loader for a description of this TV.

CIB: (the Core Image Buffer) The buffer on which most of the first 4000 words of core are saved. Although the CIB occupies two cylinders, the last two sectors are not used. See the section on the Loader for a description of the CIB and its use.

Cold Start Routine: The routine which initializes the 1130 Disk System Monitor by reading down from the disk the Skeleton Supervisor.

COMMA (the Core Communication Area): The part of core which is reserved for the work areas and parameters which are required by the Monitor programs. In general, a parameter is found in COMMA if it is required by two or more Monitor Programs or if it is passed from one Monitor Program to another. COMMA is initialized from DCOM by the Cold Start Routine and at the beginning of each JOB.

Control Record: One of the records (card or paper tape) which directs the activities of the 1130 Monitor System. For example, // DUP is a Monitor control record that directs the Monitor to initialize DUP; *DUMPLET is a DUP control record directing DUP to initialize the DUMPLET program; *EXTENDED PRECISION is a FORTRAN control record directing the compiler to allot three words instead of two for the storage of data variables.

Core Image format: Sometimes abbreviated CI format. It is the format in which whole core loads are stored on the disk prior to execution.

Core Image Header Record: A part of a core load stored in Core Image format. It is actually the last 15 words of the format. Among these 15 words are the ITV and the setting for index register 3.

Core Image program: A mainline program which has been converted, along with all of its required subroutines, to CI format. In other words, it is a core load.

Core load: Synonymous with the term object program, which is comprised of the ITV, the object-time TV, the information contained in the Core Image Header Record, the in-core code, and all LOCALs and SOCALs.

Cylinderize: The process of rounding a disk block/sector address up to the disk block/sector address of the next cylinder boundary.

Data block: A group of words consisting of a data header, data words, and Indicator Words for a routine in Disk System format. A new data block is created for every data break. (A data break occurs whenever there is an ORG, BSS, or BES statement, at the end of each record, and whenever a new sector is required to store the words comprising a routine.)

Data break: Sometimes referred to as a break in sequence. See "Data block" for a definition of this term.

Data file: An area in either the User Area or the Fixed Area in which data is stored.

Data format: The format in which a Data file is stored in either the User Area or the Fixed Area.

Data group: A group of not more than nine data words of a routine in Disk System format. In this format every such group has as its first word an associated Indicator Word. Normally a data

group consists of eight data words plus its Indicator Word; but, if the data block of which the data group is a part contains a number of data words which is not a multiple of eight, then the last data group will contain less than nine data words.

Data header: The first pair of words in a data block for a routine in Disk System format. The first word contains the loading address of the data block, the second the total number of words contained in the data block.

DCOM (the Disk Communications Area): The disk sector which contains the work areas and parameters for the Monitor Programs. It is used to initialize COMMA by the Cold Start Routine and at the beginning of each JOB (see "COMMA").

Exit control cell: The second word following an LIBF entry point, counting the entry point as one word. It is through the contents of the Exit Control Cell that the exit from the LIBF routine is made. See the section on the Loader for explanation of this cell and its contents.

Fixed area: The area on disk in which core loads and data files are stored if it is desired that they always occupy the same sectors. No routines in Disk System format may be stored in this area.

FORTRAN core load: A core load which was built from a mainline written in FORTRAN.

Hardware area: Occupies core locations 0_{10}-39_{10}. Words 1, 2, and 3 are index registers 1, 2, and 3, respectively. Words 8-13 are the Interrupt Transfer Vector (ITV). Words 32-39 are the buffer required by the 1132 Printer (words 38 and 39 are also used by the Skeleton Supervisor). Those core locations in this area which were not specifically mentioned are reserved.

IBM area: That part of disk storage which is occupied by the Monitor Programs; i.e., cylinders 0-33 (sectors 0-271).

ILS (an Interrupt Level Subroutine): A routine which services all interrupts on a given level; i.e., it determines which device on a given level caused the interrupt and branches to a servicing routine (ISS) for processing of that interrupt. After this processing is complete, control is returned to the ILS, which turns off the interrupt.

DEFINE FILE table: The table which appears on the very first sector of any mainline which refers to defined files. There is one 7-word entry for each file which has been defined.

Disk block: A 20-word segment of a disk sector. Thus, sixteen disk blocks comprise each sector. The disk block is the smallest distinguishable increment for DSF programs. Thus the Monitor System permits packing of DSF programs at smaller intervals than the hardware would otherwise allow. The disk block is also referred to elsewhere as the "disk byte".

Disk System format: Sometimes abbreviated DSF. It is the format in which mainlines and subroutines are stored on the disk as separate entities. It is not possible to execute a program in DSF; it must first be converted to Core Image format.

Disk System format program: A program which is in Disk System format. It is sometimes called a DSF program.

Effective program length: The terminal address appearing in a program. For example, in Assembler Language programs it is the last value taken on by the Location Address Counter and appears as the address assigned to the END statement.

Entry point: A term which may give rise to confusion unless the reader is careful to note the context in which this term appears. Under various conditions it is used to denote 1) the symbolic address (name) of a place at which a subroutine or a Monitor Program is entered, 2) the absolute core address at which a subroutine or mainline is to be entered, and 3) the address, relative to the address of the first word of the subroutine, at which it is to be entered.

Indicator Word: Tells which of the following data words should be incremented (relocated) when relocating a routine in Disk System format. It also tells which are the names in LIBF, CALL, and DSA statements. Routines which are in Disk System format all contain Indicator Words, preceding every eight data words. Each pair of bits in the Indicator Word is associated with one of the following data words, the first pair with the first data word, etc.

Instruction address register: Also called the I-counter. It is the register in the 1130 which contains the address of the next sequential instruction.

In-core routine: A part of a given core load which remains in core storage during the entire execution of the core load. ILSs are always in-core routines, whereas LOCALs and SOCALs never are.

ISS (an Interrupt Service Subroutine): A routine which is associated with one or more of the six levels of interrupt; i.e., CARD0, which causes interrupts on two levels, is such a routine.

ISS counter: A counter in COMMA (word 50) which is incremented by 1 upon the initiation of every I/O operation and decremented by 1 upon receipt of an I/O operation complete interrupt.

ITV (Interrupt Transfer Vector): The part of the Hardware Area which supplies the second words of the automatic BSI instructions which occur with each interrupt. In other words, if an interrupt occurs on level zero and if core location eight contains 500, an automatic BSI to core location 500 occurs. Similarly, interrupts on levels 1-5 cause BSIs to the contents of core locations 9-13, respectively. The ITV is defined as core locations 8-13.

Job: A group of tasks (subjobs) which are to be performed by the 1130 Disk Monitor System and which are interdependent; i.e., the successful execution of any given subjob (following the first one) depends upon the successful execution of at least one of those which precedes it. See the section on the Supervisor for examples.

LET/FLET (the Location Equivalence Table for the User Area/the Location Equivalence Table for the Fixed Area): The table through which the disk addresses of Programs and Data files stored in the User Area/Fixed Area may be found. LET occupies the cylinder following the Supervisor Control Record Area. If a Fixed Area has been defined, FLET occupies cylinder 34 (sectors 272-279); otherwise, there is no FLET.

LIBF routine: A routine which must be referenced with an LIBF statement. The type codes for routines in this category are 3 and 5.

LIBF TV: The transfer vector through which LIBF routines are entered at execution time. See the section on the Loader for a description of this TV.

Loading address: The address at which a routine or data block is to begin. In the latter case the address is that of an absolute core location, while in the former it is either absolute or relative, depending upon whether the routine is absolute or relocatable, respectively.

Load-time TV: The transfer vector which the Loader uses during the building of a core load. See the section on the Loader for a discussion of this TV.

LOCAL (load-on-call routine): That part of an object program which is not always in core. It is read from Working Storage into a special overlay area in core only when it is referenced in the object program. LOCALs, which are specified for any given execution by the User, are a means of gaining core storage at the expense of execution time. The Loader constructs the LOCALs and all linkages to and from them.

Location assignment counter: A counter maintained in the Assembler program for assigning addresses to the instructions it assembles.

Modified EBCDIC code: A six-bit code used internally by the Monitor programs. In converting from EBCDIC to Modified EBCDIC, the leftmost two bits are dropped.

Modified Polish Notation: The rearrangement of operators and operands (i.e., an operator and two operands) into the triple form required by the FORTRAN Compiler to generate the code necessary to perform arithmetic operations.

Monitor Program: One of the following parts of the 1130 Disk System Monitor: Supervisor (SUP), Disk Utility Program (DUP), Assembly Program (ASM), and FORTRAN Compiler (FOR).

Name code: The format in which the names of subroutines, entry points, labels, etc. are stored for use in the Monitor Programs. The name consists of five characters, terminal zeros being added if necessary to make five characters. Each character is in modified EBCDIC code, and the entire 30-bit representation is right-justified in two 16-bit words. The leftmost two bits are

used for various purposes by the Monitor Programs. In FORTRAN, symbols of one or two characters only are packed in modified EBCDIC code into a single word.

NOCAL (a load-although-not-called routine): A routine which is to be included in an object program although it is never referenced in that program by an LIBF or CALL statement. Debugging aids such as a trace routine or a dump routine fall into this category.

NOP: Used to denote the instruction, No Operation.

Object program: Synonymous with the term core load.

Object-time TV: A collection of both the LIBF TV and the CALL TV.

Principal I/O device: The 1442 Card Read/Punch if one is present; the 1134 Paper Tape Reader/ 1055 Paper Tape Punch otherwise.

Principal print device: Sometimes referred to as the Principal Printer. It is the 1132 Printer if one is present; the Console Printer otherwise.

Program header record: A part of a routine stored in Disk System format. Its contents vary with the type of the routine with which it is associated. It contains the information necessary, along with information from LET, to identify the routine, to describe its properties, and to convert it from Disk System format to a part of a core load.

Relocatable program: A program which can be executed from any core location. Such a program is stored on the disk in Disk System format.

Relocation: The process of adding a relocation factor to address constants and to those two-word instructions whose second words are not (1) invariant quantities, (2) absolute core addresses, or (3) symbols defined as absolute core addresses. The relocation factor for any program is the absolute core address at which the first word of that program is found.

Relocation indicator: The second bit in a pair of bits in an Indicator Word. If the data word with which this bit is associated is not an LIBF, CALL, or DSA name, then it indicates whether or not to increment (relocate) the data word. If the relocation indicator is set to 1, the word is to be relocated.

Resident Monitor: Occupies core locations 0_{10}-607_{10}. This area is required by the 1130 Disk Monitor System for its operation and is generally unavailable to the user for his own use. The Resident Monitor consists of the Hardware Area, COMMA, the Skeleton Supervisor, and DISK0.

Sectorize: The process of rounding a disk block address up to the disk block address of the next sector boundary.

Skeleton supervisor: That part of the Supervisor which is always in core (except during the execution of FORTRAN core loads) and which is, essentially, the logic necessary to process CALL EXIT and CALL LINK statements. Together with COMMA it occupies core locations 38_{10}-144_{10}.

SOCAL (a System Overlay to be loaded-on-call): One of three overlays automatically prepared by the Loader under certain conditions when a core load is too large to fit into core storage. See the section on the Loader for an explanation.

Subroutine: Used in the 1130 Disk Monitor System interchangeably with the term subprograms, routine, and program. Any distinctions between these terms will have to be inferred from the context.

Supervisor control record area: The area in which the Supervisor Control Records are written. This area is the cylinder following the CIB. The first two sectors are reserved for *LOCAL records, the next two for *NOCAL records and the next two for *FILES records. The last two sectors in this cylinder are not utilized. See the Supervisor section for the formats of these records.

The Monitor: Refers to the 1130 Disk System Monitor.

User area: The area on the disk in which all routines in Disk System format are found. Core loads (i.e., programs in Core Image format) and Data files may also be stored in this area. All IBM-supplied routines are found here, since they are stored in Disk System format. This area begins at the cylinder following LET and occupies as many sectors as are required to store the routines and files residing there.

User programs: Are mainlines and subroutines which have been written by the user.

User storage: That part of disk storage which is neither Working Storage nor the IBM Area. It begins at cylinder 34 (sector 272), which would be the beginning of the CIB unless a Fixed Area is defined. In this case FLET would occupy cylinder 34 (sectors 272-279), the Fixed Area would begin at cylinder 35 (sector 280), and the CIB would occupy the first two cylinders following the Fixed Area, the length of which is defined by the user.

Working storage: The area on disk immediately following the last sector occupied by the User Area. This is the only one of the three major divisions of disk storage (IBM Area, User Storage, Working Storage) which does not begin at a cylinder boundary.

XR1, XR2, XR3: The acronyms for index registers 1, 2, and 3, respectively.

INDEX

IBM 1130 Printed in U.S.A. Z26-3752-0

IBM

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

READER'S COMMENT FORM

IBM 1130 Monitor Programming System
Program Logic Manual

Form Z26-3752-0

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

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

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

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

Other _____

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

COMMENTS

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

Z26-3752-0

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

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

POSTAGE WILL BE PAID BY . . .

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

Attention: Programming Publications, Dept. 452

fold fold

IBM 1130 Printed in U.S.A. Z26-3752-0

IBM

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