File Number 1130-28
Form Y21-0010-0

Program Logic

IBM 1130 RPG

Program Number 1130-RG-007

This publication describes the internal logic of the RPG compiler for the 1130 Computing System. It is intended for use by persons involved in program maintenance, and system programmers who are altering the program design. Program logic information is not necessary for the use and operation of the program; therefore, distribution of this publication is limited to those with the aforementioned requirements.

Restricted Distribution

PREFACE

This program logic manual (PLM) supplements the program listing of the 1130 RPG Compiler (referred to in this publication as RPG) by describing the program.

The first section of this PLM starts by discussing the overall structure of the RPG compiler. Following this, each phase is described individually and is accompanied by a flowchart of the logical elements.

The second section of this PLM describes the main routines of the RPG object program. The description contains flowcharts and narrative which serve to illustrate the cycle of operations within the object program.

Prerequisites and Related Publications:

Effective use of this publication requires an understanding of the RPG language contained in the publication IBM 1130 RPG Language, Form C21-5002.

For information on the 1130 Computing System beyond the purpose of this publication, refer to the following publications:

1. IBM 1130 Disk Monitor System, Version 2: Programming and Operator's Guide, Form C26-3717.

2. IBM 1130 Disk Monitor System, Version 2: Program Logic Manual, Form Y26-3714.

3. IBM 1130 Disk Monitor System, Version 2: System Introduction, Form C26-3709.

4. IBM 1130 Subroutine Library, Form C26-5929.

For titles and abstracts of associated publications, see IBM 1130 Bibliography, Form A26-5916.

First Edition

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department 425, Rochester, Minnesota 55901.

FIGURES

TABLES

CHARTS

IBM 1130 RPG

The IBM 1130 RPG language provides an efficient technique for writing source programs that can be translated into object programs (machine language) by the 1130 RPG compiler.

1130 RPG consists of a source language and a compiler. The source language allows definitions of characteristics of the files to which the input and output records belong, the fields of input data records, the literals, the operations and calculations to be performed, and the fields of the output records. The RPG language entries specified on the RPG coding form make up the source program.

The RPG compiler reduces the input/output operations and the number of data passes to a minimum. Input/output operations are reduced by retaining as much source data as possible in main storage. All blanks, comments, and unrequired fields are deleted from the source specifications, and the resulting compressed source specifications are placed in a reserved area of main storage called a compression buffer. The term compression (or compression record), as used in this publication, refers to the data compressed from one source statement. Compression blocks refers to a group of these compressed specifications. Compression block one is variable in length depending on the amount of core storage available. All succeeding blocks are of fixed length (2560 words). Examples of the compression record formats for each specification type are included under "COMPRESSION FORMATS."

The number of iterations through compression records is reduced by placing unique field names, literals, and resulting indicators into tables. The areas allotted for the tables are large enough to contain all of the entries in most of the programs to be compiled. As a result, addresses can be assigned to the entries immediately, and machine instructions can be generated with a minimum number of iterations through compression.

SYSTEM ENVIRONMENT

Machine Requirements

Program Generation

The minimum machine requirements for generating an RPG object program are as follows:

- 1131 with 8K words of core storage
- One card-reading device
- Single Disk Storage Feature
- One printer (IBM 1132, 1403, or console printer.)

Object Program Execution

The minimum machine requirements for the execution of an RPG object program depend on the I/O configuration used:

- 1131 with 8K words of core storage and single disk storage
- Input/Output devices as required by the object program:

  IBM 1403 Printer, Model 6 or 7
  IBM 1442-5 Card Punch
  IBM 1132 Printer
  IBM 2501 Card Reader, Model A1 or A2
  IBM 1442 Card Read/Punch, Model 6 or 7

Additional Machine Features Supported

The following system features are supported for program generation:

- 1131 with 16K or 32K words of core storage
- A card-punching device if the object program is to be punched
- One or more additional IBM 2310 disk units

PROGRAM ORGANIZATION

As shown in Figure 1, the 1130 RPG compiler consists of six major components: the Resident Phase, Enter Phases, Assign Phases, Diagnostic Phases, Input/Output, and Assemble Phases.

The Resident Phase is the first phase of the compiler. Some routines within the Resident Phase remain in storage throughout compilation. These routines handle calling a phase, getting and putting a compression block, printing and reading source statements, and printing error notes.

The Enter Phases read, list, compress, and perform diagnostics on the source statements. Also, a table of filenames is created.

In the Assign Phases, addresses are assigned to Resulting Indicators, defined field names, and Calculation and Output literals in the compression. Also, a symbol table is printed.

The Diagnostic Phases detect errors not detected in the Enter Phases, list all multi-defined, undefined, and unreferenced field names, and print abbreviated error messages for all diagnostic errors that have occurred during the generation.

The I/O Phases build a table of file description entries and then used this table to produce object code for I/O requests involving certain devices and processing methods. Object code necessary to interface with ISAM subroutines is also generated.

The Assemble Phases generate most of the object program code, set up the tables used for output linkage, and generate the necessary linkage. Although included in the Assemble phases, the Terminate Compilation Phase (RG60) performs a separate compiler function: this phase terminates compilation, either naturally or due to errors in the compilation.

METHOD OF OPERATION

This section presents an overview of the main functions of the RPG compiler and the sequence of events that bring about these functions. The five main functions of the 1130 RPG compiler described in this section are:

- System initialization
- Input processing
- Diagnosing, noting, and describing errors
- Generating object code
- Final processing

To convey the logic and data flow of these functions, this section includes a series of diagrams building from the general to the specific. Supporting text is included, where necessary, but, for the most part, the diagrams are designed to be self-explanatory. The consecutive progression of events that occur within individual phases are not described here; this information is included under "PHASE DESCRIPTION".

Certain times, in the following flowcharts and text, abbreviations are used. A partial list of these abbreviations and their meanings follows:

COMMA	- Resident Monitor Communication Area
COMP	- Compression
COMAREA	- Communication Area
DSF	- Disk System Format
ES	- Extension Specification
EXT	- Extension
FDS	- File Description Specification
NEIT	- Neither (Flowchart usage)
PARAM	- Parameter
WS	- Working storage

RESIDENT ROUTINES AND THE COMMUNICATION AREA

To save load time in each phase, routines used by two or more phases (common routines) are stored in the Resident phase (RG00) and remain there as long as necessary.

Linkage to the common routines from a processing phase is accomplished through a branch within that phase. The common routine then performs the operation desired by the requesting phase and returns control to it.

The common routines originally contained in RG00 are:

- Get Compression Routine (GETCM)
- Print Error Note Routine (PRTER)
- Print Source Card Routine (PRTSP)
- Read Source Card Routine (RDSPC)
- Put Compression Routine (PUTCM)
- Call Phase Routine (CALPH)

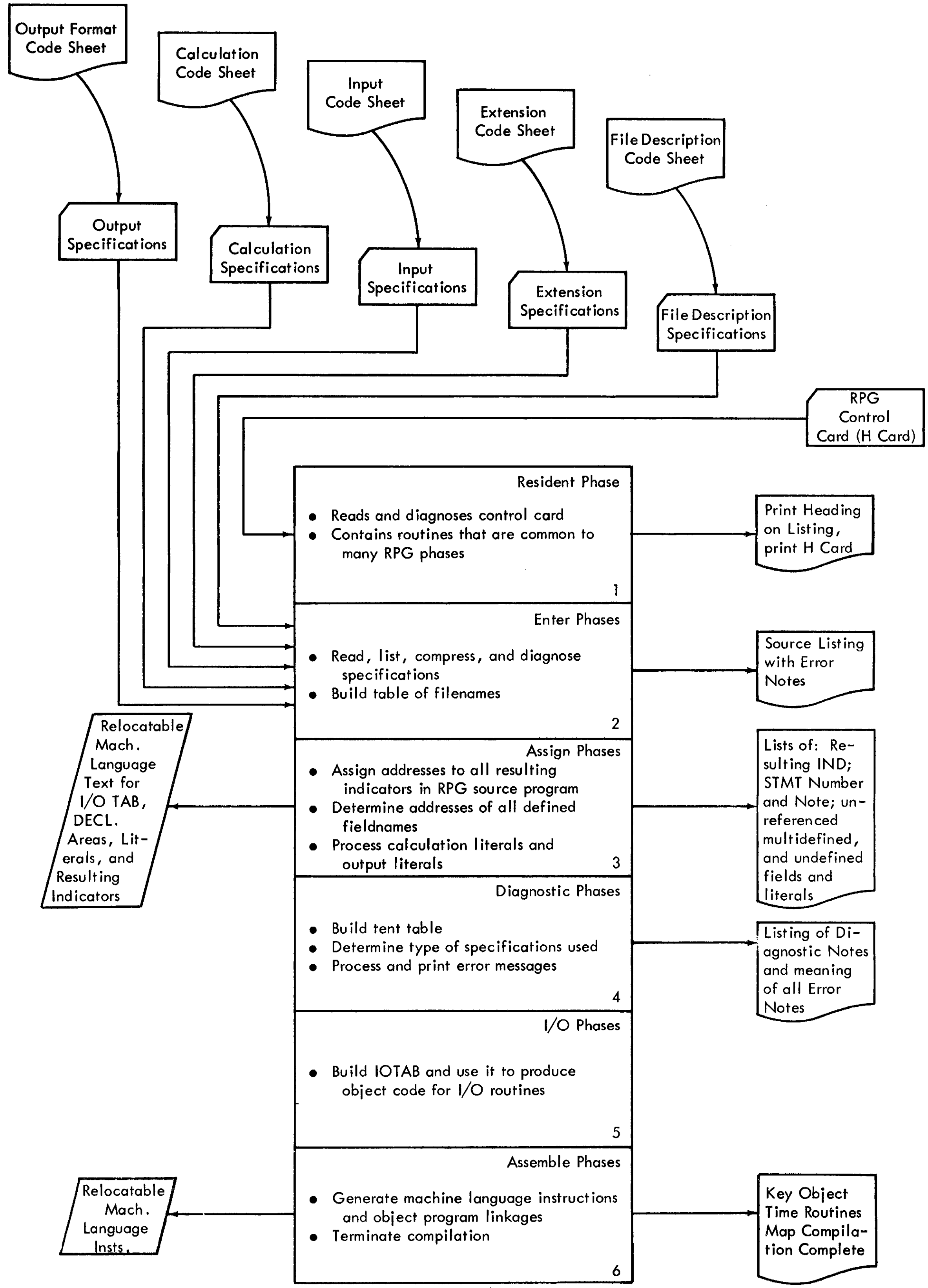


Figure 1. Program Block Diagram

Of these, PRTER and PRTSP are replaced by new routines created by phase RG10. (These new routines retain the same names and perform approximately the same functions as the original routines.)

Two other routines are moved into RG00 by phase RG10:

- Object Code Routine (PUTOB)
- Complete Object Code Routine (OBEND)

The common routines are described in detail in the "PROGRAM ORGANIZATION" section.

In addition to the common routines, a Communication Area (COMAREA) is established in the Resident phase and remains there throughout processing by the RPG compiler. The COMAREA is an 80-word communication area that contains information that must be transferred between phases of the RPG compiler. It begins at address 'ZRDSP' and includes such information as the starting address of the compression of each type of specification, addresses of routines in the Resident areas, and other constants and addresses used during compilation. The format and contents of the COMAREA are described under "CONTROL BLOCKS AND TABLES".

LINKAGE BETWEEN PHASES

Except for the Resident phase, which remains in storage throughout compilation, each RPG phase is brought into storage only when it is needed. When a phase completes processing, it returns to the CALPH routine with a request to bring another phase into storage. The CALPH routine then substitutes the requested phase for that just completed, passing control to the requested phase.

SYSTEM INITIALIZATION

Before the RPG Control Card or RPG specification cards can be read, some common routines must be read into storage and the COMAREA must be defined. Both operations are performed when the Resident phase (RG00) is given control. When the Resident phase is first given control, it contains six common routines, as well as instructions designating where these routines are to be placed. As soon as the Monitor brings this phase into storage, it relocates the common routines and passes control to the first instruction. Next, the COMAREA is defined, and the RPG Control Card is processed.

Input and output flow for the initialization function is shown in Figure 2.

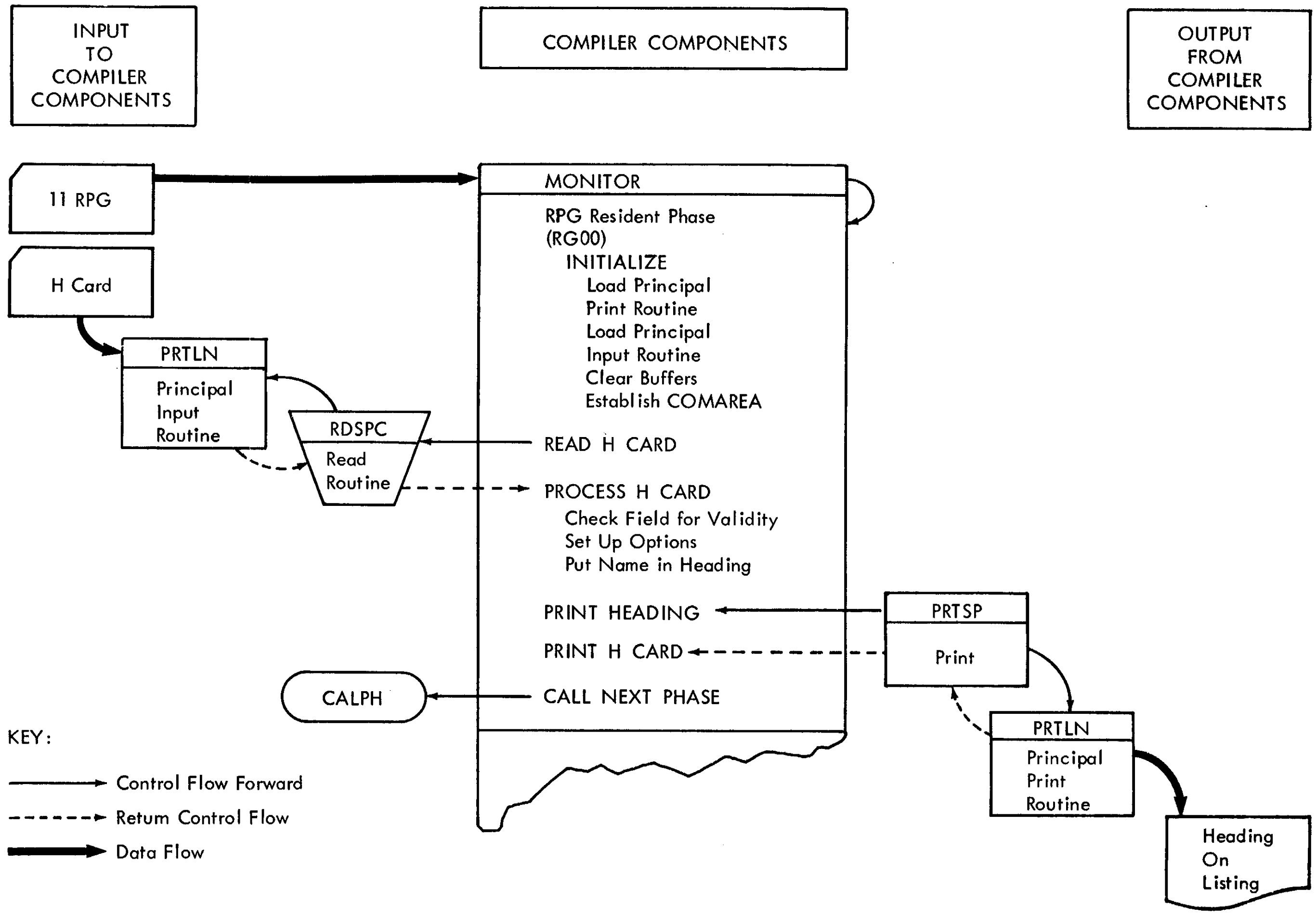


Figure 2. Initialization Functions of the RPG Compiler

INPUT PROCESSING

Information on the RPG Control Card and the specification cards must be recorded in storage so that it can be easily accessed by the Diagnostic, I/O, and Assemble phases of the compiler. This operation is performed by the Enter phases.

In addition to showing the input and output flow for the input processing function, Figure 3 depicts the data and control flow for evaluating and compressing information in the user's RPG source program.

DIAGNOSING, NOTING, AND DESCRIBING ERRORS

Diagnosis of errors is first performed in the Resident phase, when checks are made for such things as an invalid Monitor Control Card and exceeding the limits of working storage. If an error such as this is found, exit is made to RG60, where an error message is printed, and compilation is terminated.

Diagnosis of another type takes place in the Enter phases. If invalid statements are detected, they are noted, and compilation continues. This holds true for the Assign phases, where invalid indicators, fields or literals are detected and noted. If no valid input, output, or file description compressions are read by RG10, it notes the error and exits immediately to RG19. RG19, Diagnostic message phase, contains the program that prints error-note messages. Each time a phase prior to RG19 notes an error, it causes the Resident phase to print the note-number identifying the error. Then, when RG19, RG20, and RG21 gain control, error notes are printed corresponding to the error-note numbers. If a terminal error, such as no valid compression, is processed by RG19, RG20, or RG21, an exit is taken directly to RG60. At RG60, an error-note may be printed, as well as a "compilation ended" note, followed by an exit to the monitor.

The diagnosing, noting and identifying functions are illustrated by Figure 4.

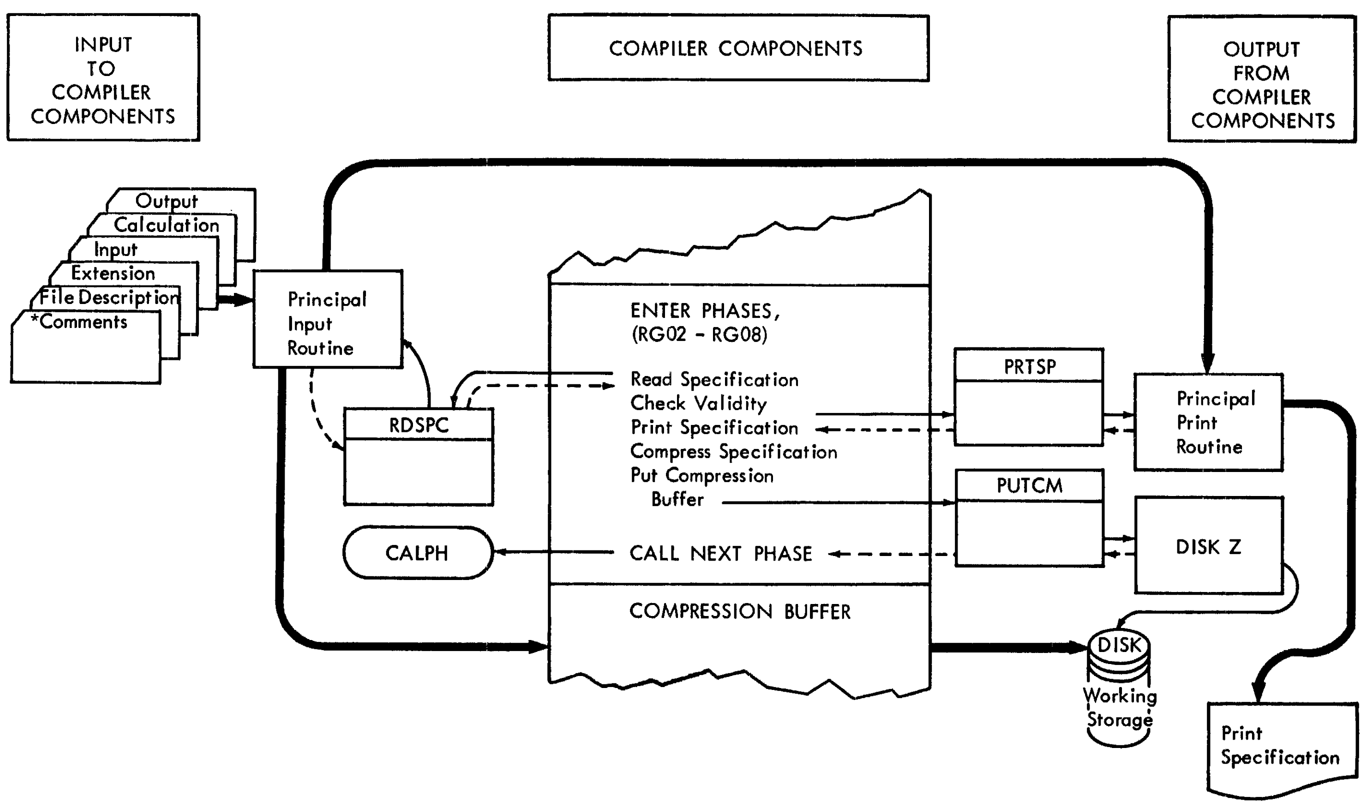


Figure 3. Input Processing Functions of the RPG Compiler

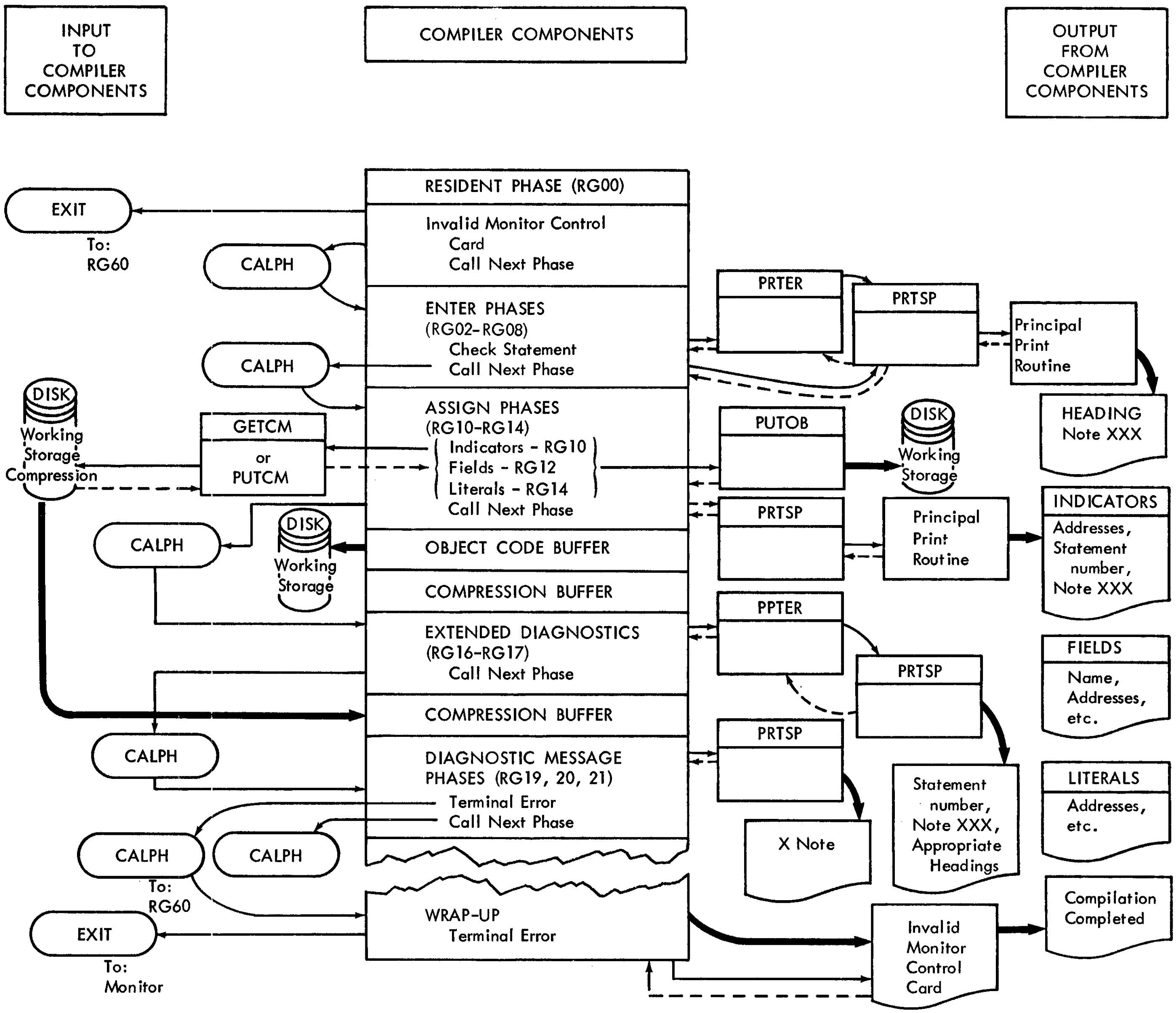


Figure 4. Diagnostic Functions of the RPG Compiler

GENERATING OBJECT CODE

One of the primary objectives of phases RG22-RG54 is to generate object code. This is accomplished by linking to PUTOB. Two addresses are passed to PUTOB: the address of the code to be generated, and the address of the table that describes the code. (Refer to PUTOB--Put Object Code.)

Depending on the type of statement processed, object code is generated for a specific function. In addition to generating code, these phases store the addresses of some of the generated routines in the NOTES section of the COMAREA. When RG58 gains control, it takes the addresses from NOTES, generates them into the proper order in the fixed driver (see page 160), and places the object code in the object code buffer. At the same time it prints the Key Addresses of object program map, a listing of routine names and addresses. The generation of object code is shown in Figure 5.

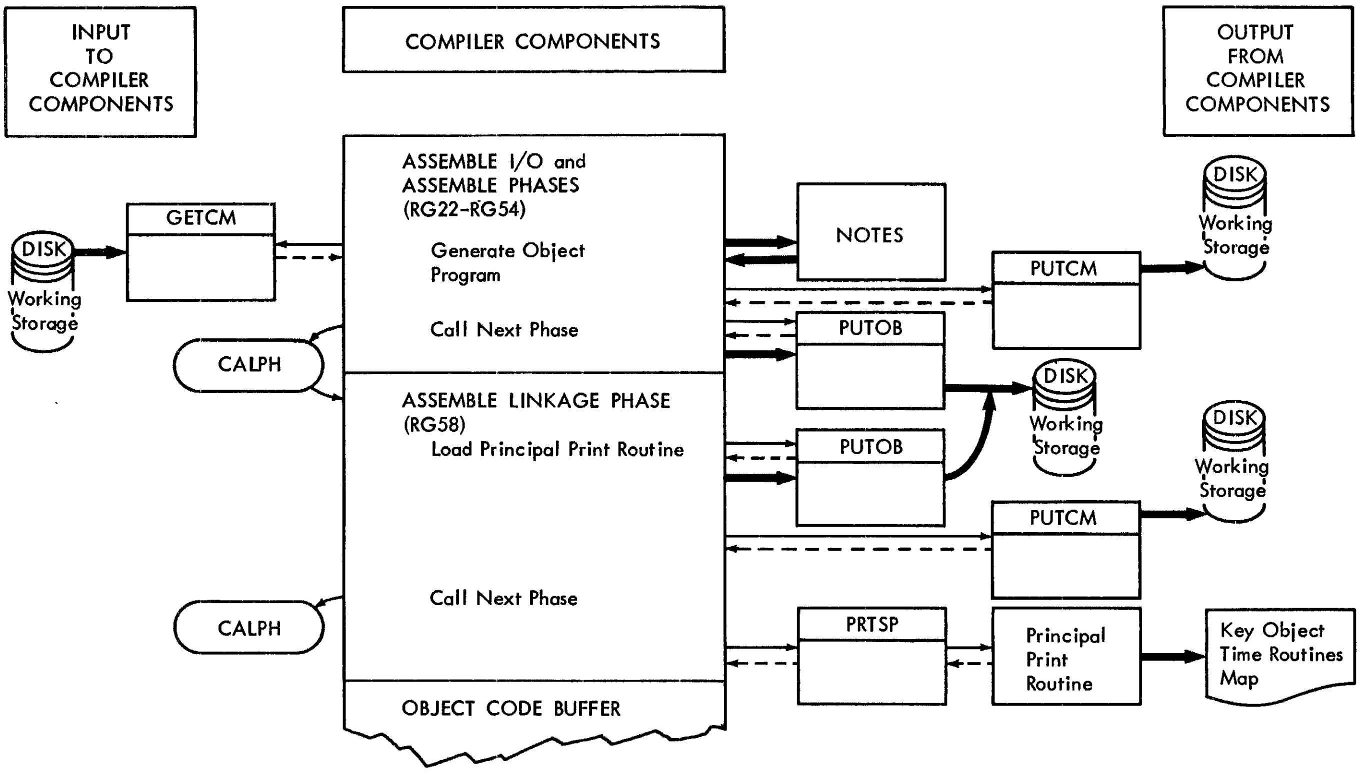


Figure 5. Generate Object Code Functions of the RPG Compiler

FINAL PROCESSING

Final processing takes place when RG60 gains control. Any generated object code remaining in the Object Code Buffer is moved to disk working storage. If necessary, the object code is then read into the Disk I/O Buffer and moved to the beginning of working storage.

After the object code is moved, the DUP and EXEC switches in the Monitor Communication Area (COMMA) are set, and such values as block count and relative entry point are entered in the Disk Communication Area (DCOM). Then, a "compilation complete" message is printed and exit is made to the monitor.

Final processing is depicted in Figure 6.

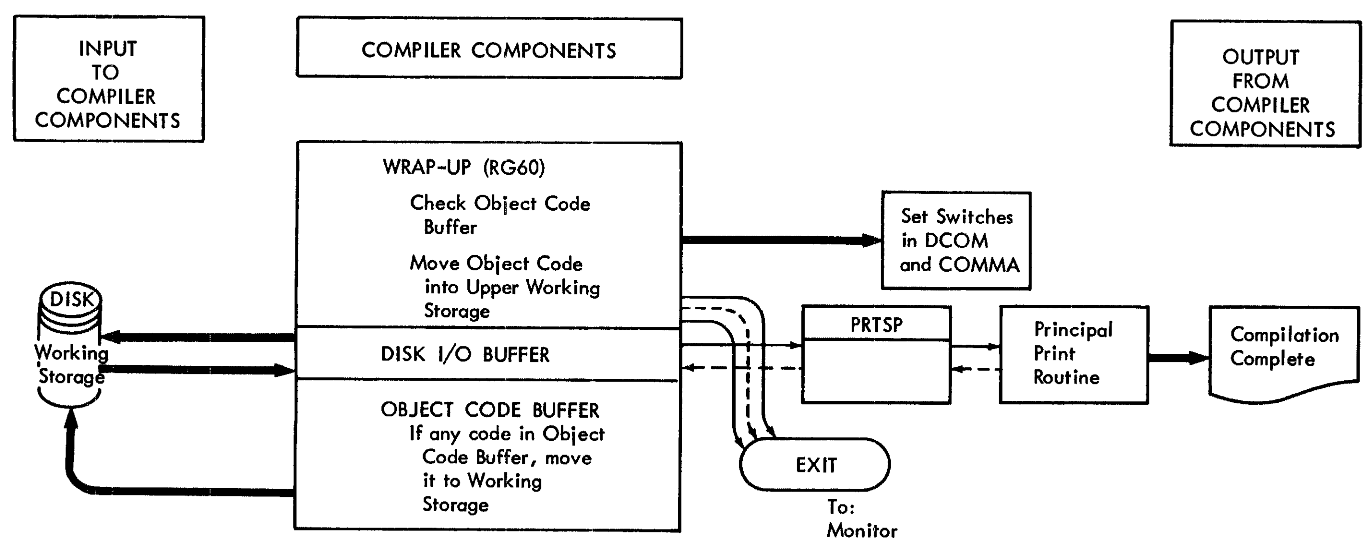


Figure 6. Final Processing Functions of the RPG Compiler

This section describes the design of the RPG compiler and describes how the program is packaged.

FUNCTIONAL ORGANIZATION

As mentioned in Section 1, the six major components of the 1130 RPG Compiler are the Resident Phase, Enter Phases, Assign Phases, Diagnostic Phases, I/O Phases, and Assemble Phases.

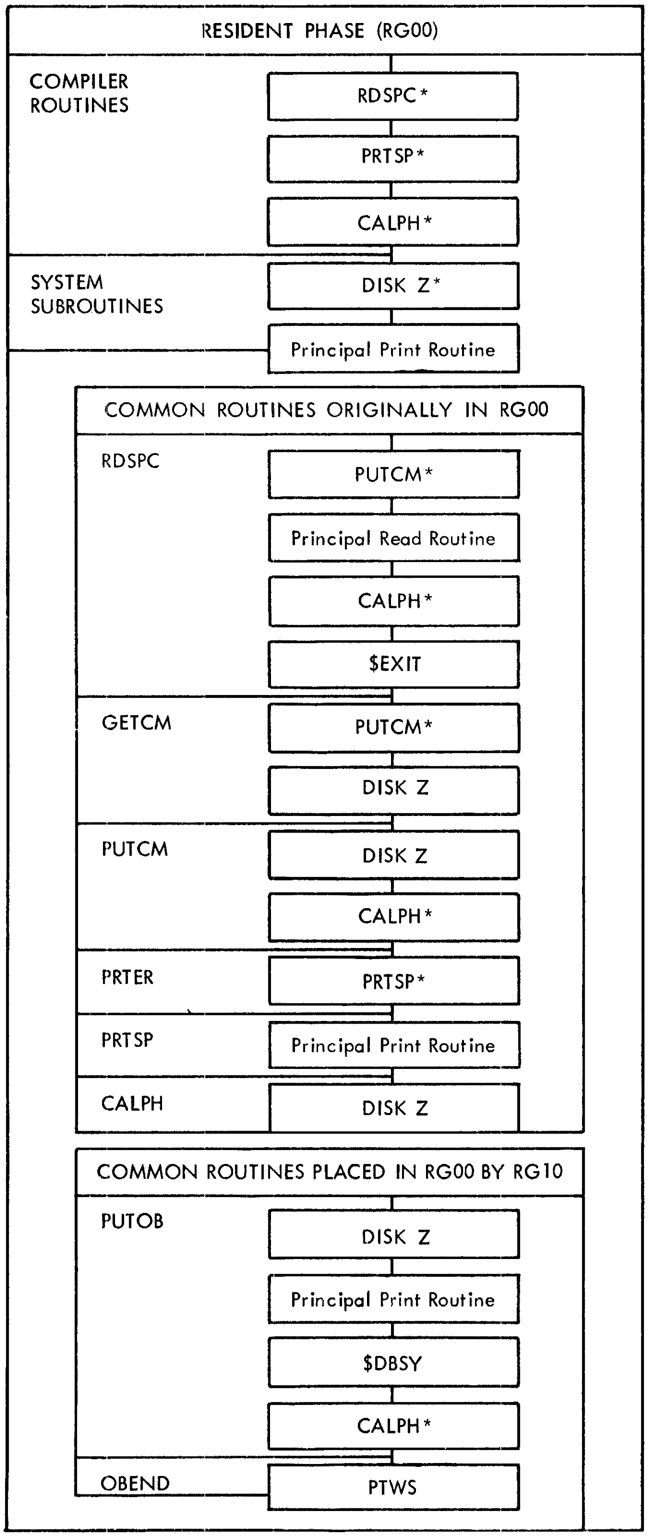
Resident Phase

The Resident phase of the RPG Compiler remains resident in the same position throughout compilation as shown by Table 1. This phase is composed mainly of code, which accomplishes the following functions:

- Fetch and store the principal print, input, and conversion routines.
- Read first source card.
- Initialize compression buffer.
- Process and diagnose header card.
- Print compiler listings, if required.
- Print header card and error notes.
- Read a card to ready the input buffer for the first Enter phase.
- Call the first Enter phase.

Also included in the Resident phase is a Communication Area (the COMAREA). This area provides addresses and constants used by the compiler. (The COMAREA is described in detail in "Table 3: Resident Communications Area".

Also stored in the Resident phase are six common routines, which are used by more than one phase. (The first Assign phase (RG10) replaces two of these routines, and builds two other common routines, which are also stored in the Resident phase, after it is expanded to accommodate them.) Figure 7 shows the use of routines by the Resident phase and by the common routines (which are contained within the Resident Phase).



**The use of the routine is shown in Figure 7, under Common Routines.

Please note: PRTER and PRTSP are overlaid by RG10 with routines bearing the same addresses.

Figure 7. Resident Phase, External Routine Usage

Enter Phases

The Enter Phases, as a whole, perform the following functions: read, list, diagnose, and compress File Description, Extension, Input, Calculation, and Output-Format Specifications, and build the Filename Table. The Enter phases (and their module names) are:

- Enter File Specifications (RG02)
- Enter Input Specifications (RG04)
- Enter Calculation Specifications (RG06)
- Enter Output-Format Specifications (RG08)

The core storage layout for the Enter phases is shown in Table 1, while Figure 8 shows the use of routines by these phases.

Assign Phases

The Assign phases of the RPG compiler primarily perform the following functions: assign addresses to all resulting indicators and defined field names, print out a symbol table, build a table of indicators, and process calculation and output literals.

The Assign phases (and their module names) are:

- Assign Indicators (RG10)
- Assign Field Names (RG12)
- Assign Literals (RG14)

The core storage layout for these phases is shown in Table 1; Figure 9 shows the use of routines by each phase.

Diagnostic Phases

The Diagnostic phases of the RPG compiler perform the following functions: detect errors not found by the Enter phases; list all multi-defined, undefined, and unreferenced field names; check for errors in the specifications; and print error messages for all errors discovered by these and earlier phases.

The Diagnostic phases (and their module names) are:

- Extended Diagnostics 1 Phase (RG16)
- Extended Diagnostics 2 Phase (RG17)
- Error Message Phases (RG19, RG20, RG21)

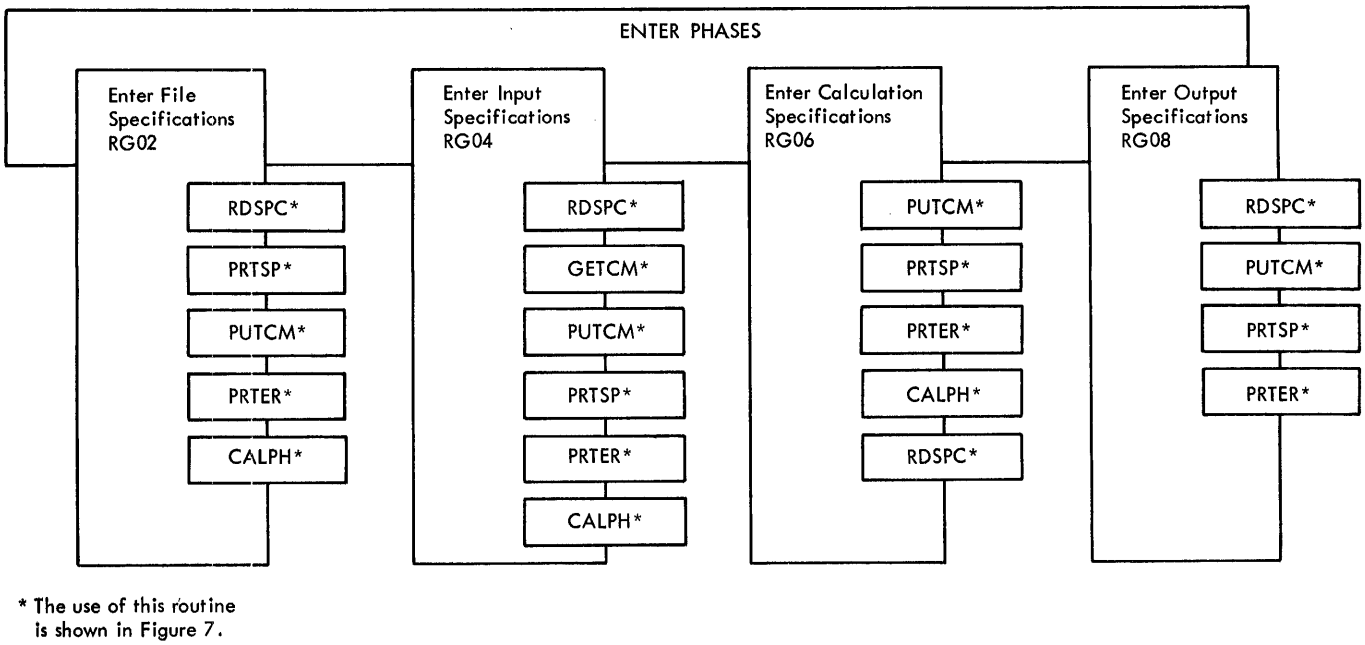


Figure 8. Enter Phases, External Routine Usage

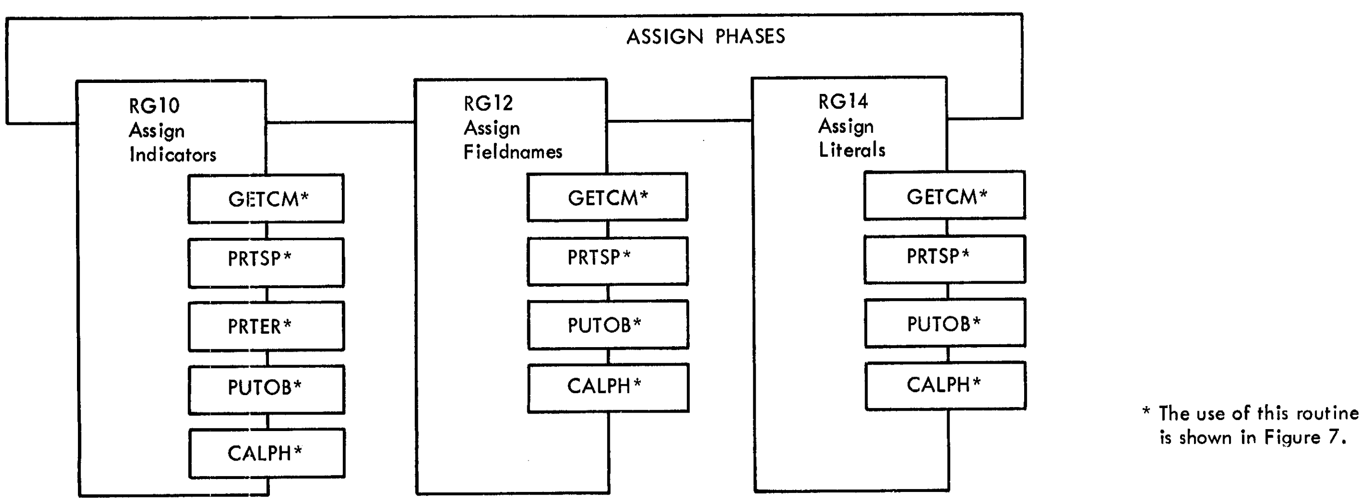


Figure 9. Assign Phases, External Routine Usage

The core storage layout for these phases is the same as for the Assign phases, as shown by Table 1; Figure 10 shows the use of routine of the Diagnostic phases.

I/O Phases

The main function of the I/O phases is to build a table of file description entries (the IOTAB) and use this table to produce object code. The I/O phases (and their module names) are:

- Assemble 1 I/O (RG22)
- Assemble 2 I/O (RG24)
- Assemble 3 I/O (RG26)
- Assemble 4 I/O (RG28)

The core storage layout for these phases is shown in Table 1.

Figure 11 shows the use of routines of these phases.

Assemble Phases

The Assemble phases generate the following: table loading and dumping routines; object code for RA and CHAIN files; object code for field type and record type input specifications; object code needed to process multiple input files; a File Input Table entry for each record type; table lookup routines; object code for calculation operations; object code to place output fields within their associated output I/O areas; object code to produce output records, and linkage from the object code

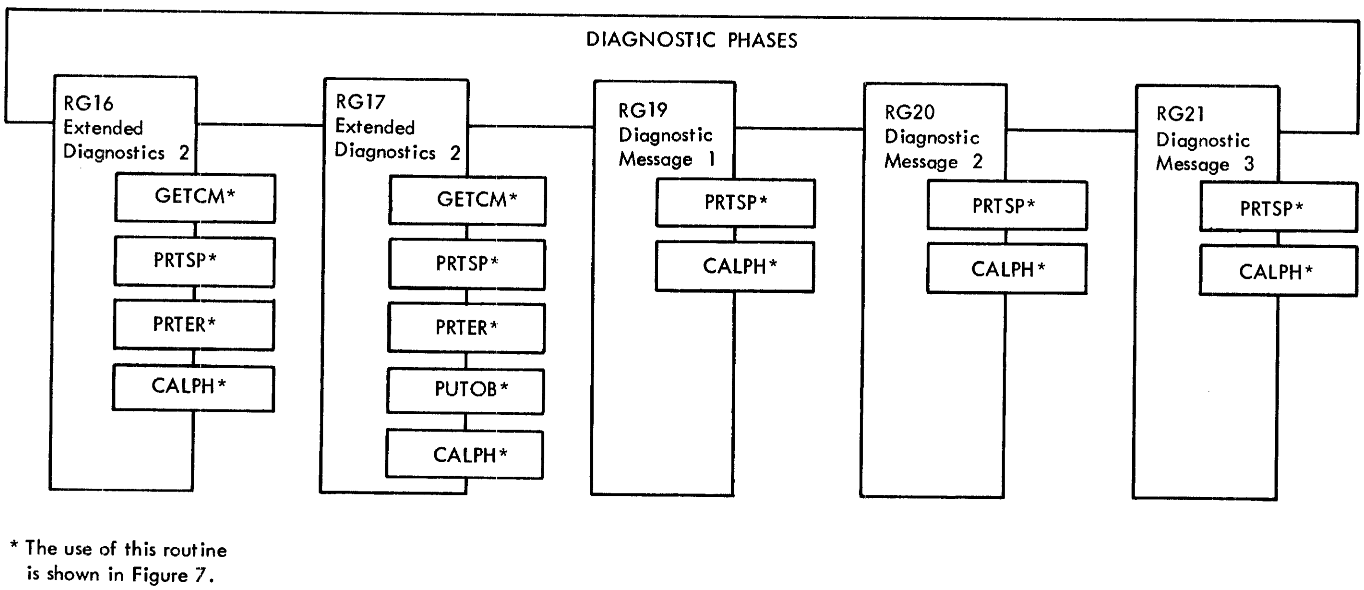


Figure 10. Diagnostic Phases, External Routine Usage

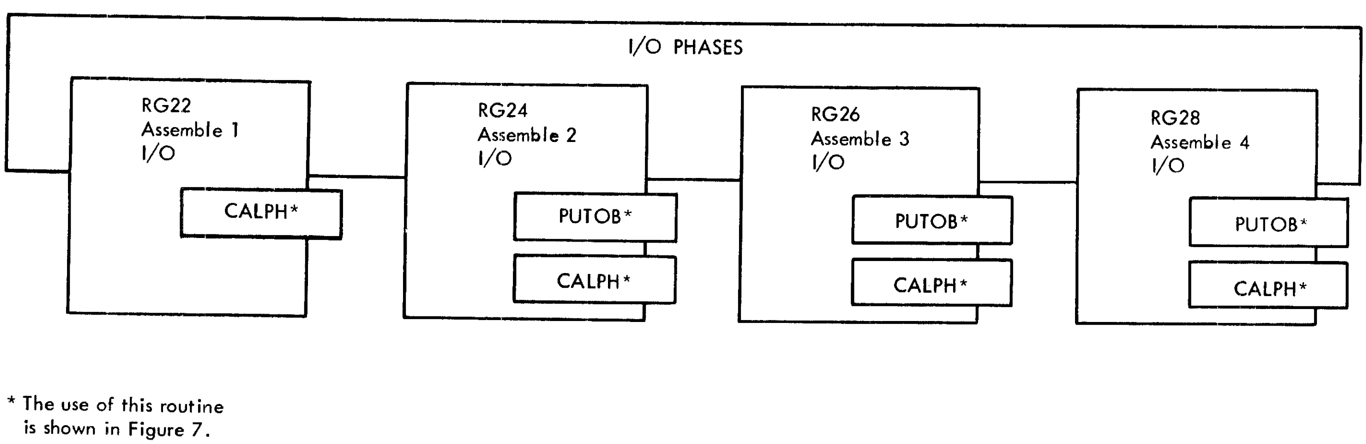


Figure 11. Input/Output Phases, External Routine Usage

to the object program. The Assemble phases (and their module names) are:

- Assemble Tables (RG32)
- Assemble Chain and RA Files (RG34)
- Assemble Input Fields (RG36)
- Assemble Control Levels (RG38)
- Assemble Multi-Files (RG40)
- Assemble Get (RG42)
- Assemble Calculation 1 (RG44)
- Assemble Calculation 2 (RG46)
- Assemble Output Fields (RG52)
- Assemble Put (RG54)
- Assemble Linkage (RG58)
- Terminate Compilation (RG60)

(Although not technically an Assemble phase, RG60 is included with these phases.)

The core storage layouts of the Assemble phases (RG22-RG54), Assemble Linkage phase (RG58), and Terminate Compilation phase (RG60) are shown in Table 1; Figure 12 shows the use of routines by each of the Assemble phases.

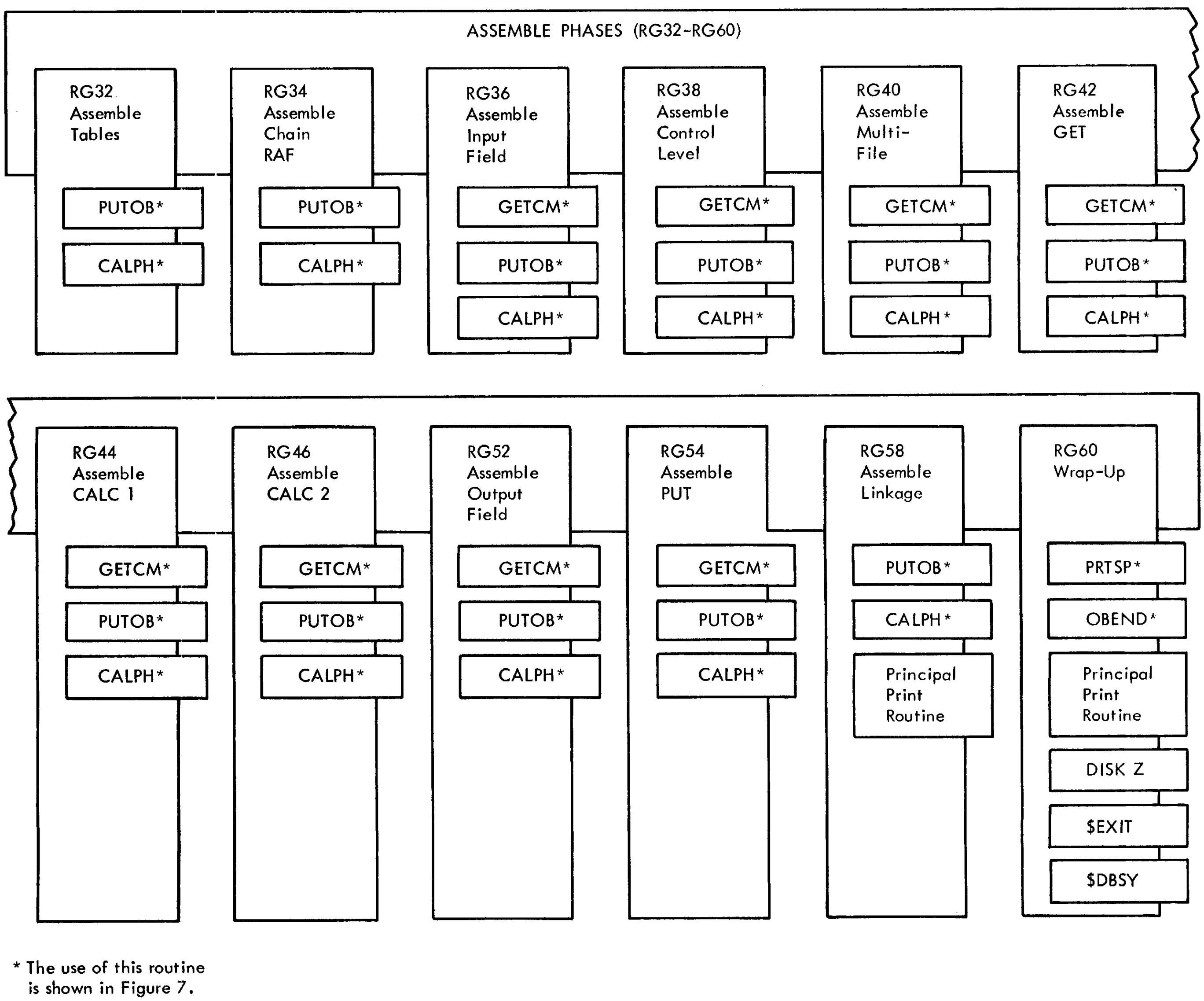


Figure 12. Assemble Phases, External Routine Usage

Table 1 shows the contents of the core storage areas in the 1130 RPG Compiler through various stages of compilation (when an 1131 with 8K words of core storage is used). In this table "Location" refers to the displacement addresses (in words) of the core storage areas.

To avoid confusion please note that the term sequence number (or internal sequence number) refers to a number assigned to each specification. This number beginning with the first specification (one) is incremented by one for each successive specification; record sequence refers to the sequence assigned to an Input record type, columns 15-16 of the Input specifications.

+Location Hex	+Location Dec	Enter	Assign and Diagnostics	I/O and Assemble	Assemble Linkage	Terminate Compilation
0	0	Resident Monitor	Resident Monitor	Resident Monitor	Resident Monitor	Resident Monitor
212	530	Principal Print Routine	Principal Print Routine	Tables	Principal Print Routine	Principal Print Routine
3C0	960	RPG Resident Phase (RG00)	RPG Resident Phase (Common Routines) ***	RPG Resident Phase (Common Routines)	RPG Resident Phase (Common Routines)	RPG Resident Phase (Common Routines)
565	1381	Principal Input Routine				
73A	1850		Phases RG10-RG21	Phases RG22-RG54	Phase RG58	Phase RG60 Disk I/O Buffer
906	2310	Phases RG02-RG08 (Consecutive phases overlay one another)				
AF0	2800					
D6C	3420					
1086	4230		Object Code Buffer	Object Code Buffer	Object Code Buffer	
1446	5190*	Compression Buffer 1	Compression Buffer 1	Compression Buffer 1	Compression Buffer 1	
15FE	5630**	Compression Buffer 2	Compression Buffer 2	Compression Buffer 2	Compression Buffer 2	
2000	8192					

* When an 1131 with 16K or 32K words of core storage is used, this is incremented to 7470.

** When an 1131 with 16K words of core storage is used, this is incremented to 13,630; when an 1131 with 32K words of core storage is used, this is incremented to 29,630.

*** RG10 is origined at 1190 for placement of additional common routines.

Note: The Assemble Linkage phase (RG58) and Terminate Compilation phase (RG60) are pictured separately to show the differences in their core layout as compared to the other assemble phases.

Table 1. Storage Layout

Each of the 29 phases in the compiler (and each major routine within these phases) is described by the following entries:

- Chart - Identifies the flowchart that describes the logic flow of the phase or routine. (The flowcharts are included as a group, beginning with Chart AA.)
- Functions - Describes the purpose and principal operations of the phase.
- Entry - Names the label of the first executable statement in a phase or routine.
- Input - Describes data to be processed by a phase or routine.
- Output - Describes the data which has been processed by a phase or routine.
- External References - Refer to Table 4, which shows the subroutines, constants, and addresses referenced by each phase of the RPG compiler.
- Exit - Identifies the phase which will be put in control following the current phase. The entry is further divided into normal and error exits, to cover all possible results. A normal exit calls a succeeding phase to continue compilation; an error exit calls a phase to note an error or to print an error message, and may possibly terminate compilation.
- Tables/Work Areas - Describes tables and work areas that are built or modified by each phase.

RESIDENT PHASE (RG00)

Chart: AA

Functions:

- Loads the principal input, principal input conversion, and principal print routines for use by compiler.
- Loads the interrupt transfer addresses necessary for these subroutines.
- Provides a communication area which can contain addresses and constants (COMAREA), and fills in certain addresses and constants in this area.
- Provides routines to perform input/output needed by the other phases. (See "COMMON ROUTINES".)
- Prints compilation headings, if requested.

Entry: RPG - entered from the monitor, to begin compilation.

Input: Input is via RPG source statements entered through the principal input device.

Output: A printed listing of the headings and RPG control card if requested.

External References: Refer to Table 4.

Exits:

- Normal: To RG02, via CALPH (Call Next Phase routine).
- Error: None. (Refer to COMMON ROUTINES for error exits within the common routines.)

Tables/Work Areas: ILS4, Interrupt Branch Table, is located in RG00. This table is described in the publication, IBM 1130 Disk Monitor System, Version 2, Programming and Operator's Guide, (Form C26-3717).

COMMON ROUTINES

Common routines are used by one or more phases other than the phase which built them. The six common routines originally stored in RG00 are:

CALPH, which calls a succeeding phase.
PUTCM, which puts a compression block in working storage.
PRTER, which prints error notes when an error is encountered during compilation.
GETCM, which reads a compression block from working storage to compression.
RDSPC, which reads information from cards into the input/output area.
PRTSP, which builds the I/O buffer and prints a listing if requested.

CALPH - Call Phase Routine

Chart: None.

Functions: Reads next phase from disk.

Entry: CALPH, entered when the requested phase is to be read into main storage and when control is to be transferred to the calling routine.

Output: None.

External References: Refer to Table 4.

Exit:

- Normal: To next phase.
- Error: None.

GETCM - Get Compression Routine

Chart: None

Functions:

- Reads a requested compression block into core storage.
- If requested block is already in core, returns to calling phase immediately.
- If requested block is not in core, the block presently in core is written out in working storage before the requested block is read in from working storage.

Entry: GETCM, entered when a compression block is to be read from working storage.

Input: A compression block number, as requested by a calling phase.

Output: The requested compression block.

External References: Refer to Table 4.

Exit:

- Normal: Return to calling routine or phase.
- Error: None.

PRTER - Error Note Routine

Chart: None.

Functions:

- Builds I/O Buffer for error notes.
- Posts error number in NOTES (within COMAREA) for RG19.

Entry: PRTER, entered when an error note is to be printed.

Input: Error number, from calling phase.

Output: Printed error note within I/O Buffer, and return address.

External References: Refer to Table 4.

Exit:

- Normal: To PRTSP, for actual output function.
- Error: None.

PRTSP - Print Listing Routine

Chart: None

Functions:

- Checks if List Option is on.
- Checks for indication of error in source card, if List Option is off.
- Sequence checks cards.
- Builds Input/Output Buffer.
- Prints Listing.

Entry: PRTSP, entered when a source card is to be printed.

Input: Source card.

Output: Printed Listing.

External References: Refer to Table 4.

Exit:

- Normal: Return to calling routine.
- Error: None.

PUTCM - Put Compression Routine

Chart: None.

Functions:

- Checks if compression block number greater than one; block one always remains in core, all others written in working storage.
- Checks working storage and sets error code if exceeded.
- Writes block on working storage from compression buffer.

Entry: PUTCM, entered when a compression block is to be written in working storage.

Input: Compression buffer.

Output: Compression blocks.

External References: Refer to Table 4.

Exit:

- Normal: Return to calling routine.
- Error: To Monitor (EXIT), if working storage is exceeded.

RDSPC - Get Source Routine

Chart: None.

Functions:

- Checks if monitor control record (116) is read, and halts compilation if it is read.
- Reads source card using two Input/Output areas.
- Converts I/O Buffer to unpacked EBCDIC.

Entry: RDSPC, entered when a source card is to be read.

Input: Source cards.

Output: Converted source card in the I/O buffer.

External References: Refer to Table 4.

Exit:

- Normal: To calling phase; to RG10, after last card (/*).
- Error: To Monitor (EXIT) if Monitor control card read.

Four common routines are built by RG10 and stored in RG00: PUTOB, OBEND, PRTER, and PRTSP. The two latter routines replace routines of the same names, which were built by RG00.

PUTOB - Put Object Code

Chart: None.

Function: Converts object code into DSF and puts it to disk working storage.

Entry: PUTOB.

Input: Address of the Object Code instructions to be generated, and the address of the table describing the Object Code instructions.

The table has the following format:

```
TABLE DC N          Number of words fol-
      DC M1+K1      lowing in table
      DC M2+K2
      .
      .
      DC MN+KN
```

M occupies bits 0-7 and defines the type of code:

hex 00 for absolute word,
hex 04 for relocatable word,
hex 08 for LIBF,
hex 0C for CALL,
hex 10 for DSA.

K occupies bits 8-15 and defines the number of words with the attribute M.

Output: DSF code on disk working storage.

External References: Refer to Table 4.

Exit:

- Normal: To calling phase.
- Error: To RG60, via CALPH, if disk overflows.

Tables/Work Areas: INDEX - A table of words built and used by PUTOB.

OBEND - Complete Object Code

Chart: None

Functions:

- Puts out end of program data header for core load builder.
- Writes last of object code to working storage.
- Computes disk block count of object program.

Entry: OBEND, called from RG60 via ZBLCT.

Input: None.

Output:

- Disk System Format (DSF) code for last block to working storage.
- Block count in ZBLCT in COMAREA.

External References: Refer to Table 4.

Exit:

- Normal: Return to RG60 (to GO).
- Error: To RG60, via CALPH.

ENTER FILE SPECIFICATIONS (RG02)

Chart: BA.

Functions:

- Reads, analyzes, lists and compresses entries on File Description and Extension specifications.
- Builds the Filename Table.
- Builds the File Description and Extension compression areas.
- Identifies errors found in a statement.

Entry: BEGIN, from the Resident phase.

Input: File description and Extension Specifications.

Output: A list of the RPG statements processed by this phase, and the error numbers that identify errors found on each statement. These error numbers and their meanings are described in the publication, IBM 1130 Disk Monitor System, Version 2: Programming and Operator's Guide, Form C26-3717.

Also, compressed versions of the File Description and Extension Specifications are contained in the File Description and Extension compression areas.

External References: See Table 4.

Exit:

- Normal: To RG04, Enter Input Specification.
- Error: None.

Tables/Work Areas:

- Filename Table (see Table 5. CONTROL BLOCKS AND TABLES).
- Error Note Table (Table 3, Part 6), which posts any errors connected with that specification for RG19.

ENTER INPUT SPECIFICATIONS (RG04)

Chart: BB

Functions:

- Sets pointers in the COMAREA, reads, analyzes, lists, and processes Input Specifications, creating compression records for use by later phases.
- Checks record type specifications (AND, OR, S, F) and diagnoses terminal errors.
- Checks field type input specifications, diagnoses errors, and processes entries.

Entry: BEGIN, from RG02 (Enter File Specifications).

Input: Input Specifications.

Output: A printed list of: the RPG statements processed by this phase, error numbers identifying the errors found on each statement, and compressed Input Specification records.

External References: See Table 4.

Exit:

- Normal: To RG06, if a Calculation Specification is encountered. To RG08, if an Output-Format Specification is encountered.
- Error: None.

Tables/Work Areas: FNTAD - Address of Filename table (FLENM). (See CONTROL BLOCKS AND TABLES).

ENTER CALCULATION SPECIFICATIONS (RG06)

Chart: BC

Functions:

- Reads, diagnoses, lists, and compresses Calculation Specifications.
- Builds table of valid operations.

Entry: BEG, from RG04 (Enter Input Specifications).

Input: Calculation Specifications.

Output: A list of the RPG statements processed by this phase, error numbers identifying errors found on each statement, and compressed Calculation Specification records.

External References: See Table 4.

Exit:

- Normal: to RG08, after last Calculation Specification has been processed.
- Error: None.

Tables/Work Areas: Table of valid calculations and corresponding attributes.

ENTER OUTPUT SPECIFICATIONS (RG08)

Chart: BD

Functions:

- Reads, diagnoses, lists, and compresses Output-Format Specifications. These specifications define the characteristics and fields of the data records that are to be written on the output files at object time.
- Determines if the specification defines a record type or a field of an output record.

Entry: RPG, from RG04 or RG06.

Input: Output-Format Specifications from source cards.

Output: A list of the RPG statements processed by this phase, error numbers identifying errors found on each statement, and compressed Output-Format Specification records.

External References: See Table 4.

Exit:

- Normal: To RG10 (Assign Indicators phase) after last Output-Format Specification is processed, via the CALPH subroutine.
- Error: None.

Tables/Work Areas: GABLE - table of special characters found in edit words.

ASSIGN INDICATORS PHASE (RG10)

Chart: CA

Functions:

- Scans Compression.
- Builds TABAR, an indicator table.
- Replaces Resulting indicators in compression with assigned addresses.
- Places Put Object Code routine (PUTOB) in Resident Phase (RG00).
- Overlays the Print Error routine (PRTER) and Print Listing routine (PRTSP) in the Resident phase with comparable routines which retain the same addresses (ZPTER and ZPTSP) and which will remain in RG00 for the rest of the compilation.
- Places OBEND (a wrap-up routine called by RG60) in Resident phase storage.

Entry: BEGIN, from RG08 (Enter Output Specifications phase).

Input: Compression built by Enter phases.

Output: A list of the indicators and their addresses, and error numbers identifying errors found.

External References: See Table 4.

Exits:

- Normal: To RG12 (Assign Fields phase).
- Error: To RG19 (Error Message 1 phase), if unusable compression is found, or if no input Resulting Indicator is specified.

Tables/Work Areas: TABAR - table of indicators.

ASSIGN FIELD NAMES PHASE (RG12)

Chart: CB

Functions:

- Builds table of field names, ASNFL, from names in compression.
- Assigns addresses to each field.
- Replaces fields in compression with appropriate address.

Entry: BEGIN, from RG10.

Input: Compression built by Enter phases.

Output:

- A list of all specified field names with addresses, types, lengths, and decimal positions.
- Updated compression.

External References: See Table 4.

Exits:

- Normal: To RG14, if literals were specified. To RG16, if no literals were specified.
- Error: None.

Tables/Work Areas: ASNFL - Table of field names.

ASSIGN LITERALS PHASE (RG14)

Chart: CC.

Functions:

- Assigns object addresses to all literals, constants, and edit words.
- Prints and puts all unique literals as object code.
- Builds edit words from edit codes.

Entry: BEGIN, from RG12 if literals were specified.

Input: Calculation and Output compression.

Output:

- Literals on the principal print device.
- Literals in Disk System Format for the object time code.

External References: See Table 4.

Exits:

- Normal: To Extended Diagnostics Phase (RG16).
- Error: None.

Tables/Work Areas: Table of unique literals.

EXTENDED DIAGNOSTICS 1 PHASE (RG16)

Chart: DA

Functions:

- Builds TENT table from contents of File Description Specification compression (see COMPRESSION FORMATS).
- Distinguishes File Extension Specifications as to unreferenced table name for table file and diagnoses each for validity.
- Further distinguishes the type of Input Specifications (record type or field type) and diagnoses the contents of either for validity, compatibility to each other, and compatibility between record types.
- Passes on the lengths of the chaining fields, matching fields, and control levels to the Assemble phases placing them in resident area storage.

Entry: START, from RG14 if literals were used; from RG12 if no literals were used.

Output: Heading line with error notes.

External References: See Table 4.

Exits:

- Normal: To RG17 (Extended Diagnostics 2).
- Error: None.

Tables/Work Areas:

- TENT - Built from File Description Specifications. See COMPRESSION FORMATS.
- ERTAB - Table of Error indicators.
- NOTAB - Table of Error-note numbers.

EXTENDED DIAGNOSTICS 2 PHASE (RG17)

Chart: DB

Functions:

- Further diagnoses Calculation and Output Specifications.
- Updates the table address in the Calculation Specifications to the table element hold area.
- Checks Stacker Select, Space, and Skip entries.
- Prints error-notes for undefined fields.
- Checks page field for numeric field.
- Checks if End Position is within the record.
- Checks the validity of edit words.
- Checks if edited fields are numeric.
- Puts out control level hold areas.

Entry: BEGIN, from RG16.

Input: Calculation and Output compression.

Output:

- Error-notes, on the principal print device.
- Control level hold areas and Control Level Address Table.

External References: See Table 4.

Exits:

- Normal: To RG19, Error Message Phase.
- Error: None.

Tables/Work Areas: This phase references the TENT table in RG16.

ERROR MESSAGE PHASES (RG19, RG20, RG21)

Chart: DC

Functions: Diagnoses error bits and prints error messages.

Entry: START, from RG10 (Assign Indicators phase) or from RG17 (Extended Diagnostics 2 phase).

Output: Listing of flagged error messages in Diagnostics 1, 2, and 3.

External References: See Table 4.

Exits:

- Normal: To RG22 (Assemble 1 I/O phase) if no terminating errors are found.
- Error: To RG60 (Terminate Compilation phase) if terminating errors are found.

Tables/Work Areas: None.

ASSEMBLE 1 I/O PHASE (RG22)

Chart: EA

Functions: Builds Input/Output Table from Filename table (FLENM), FILE1 Table from compression, and Overflow Table. (These tables are explained under "Control Blocks and Tables".)

Entry: BEG, from RG19, RG20, or RG21.

Input: File Description compression and Filename Table.

Output:

- A table (IOTAB) entry for each file.
- A FILE1 table entry for each file.
- Overflow Table entry for each printer file.
- ISAM Load table entry for each ISAM load file.

External References: See Table 4.

Exits:

- Normal: To RG24 (Assemble 2 I/O) after the last File Description compression specification is processed, if non-disk file; to RG26 (Assemble 3 I/O), if all files are disk files.
- Error: None.

Tables/Work Areas:

- Builds Input/Output Table from entries in the Filename table.
- Builds FILE1 Table from File Description compression (File Description Specification compression is described in Section 5).
- Builds Overflow Table.
- Builds ISAM LOAD Table.

ASSEMBLE 2 I/O PHASE (RG24)

Chart: EB

Functions: Uses IOTAB built by RG22 to produce object code for Input/Output requests of non-disk files.

Entry: BEG, from RG22.

Input: IOTAB built in RG22.

Output:

- The output of this operation is assembled routines for each non-disk file (obtained from IOTAB).
- The address of an I/O routine is saved in the first word of an entry corresponding to the IOTAB entry. The output of this operation is the first word of the FILE1 table, occupied by the routine address.

External References: See Table 4.

Exit:

- Normal: To RG26, when the last non-disk I/O routine has been assembled, or to RG32.
- Error: None.

Tables/Work Areas: IOTAB and FILE1 Table (both in RG22) are referenced.

ASSEMBLE 3 I/O PHASE (RG26)

Chart: EC

Functions:

- Uses the IOTAB built by RG22 to produce object code for I/O requests of all Sequential Disk files.
- Provides I/O areas for each file.

Entry: BEGIN, from RG24.

Input: None.

Output: Object code in Disk System Format (DSF).

External References: See Table 4.

Exits:

- Normal: To RG28 (Assemble 4 I/O Phase).
- Error: None.

Tables/Work Areas: IOTAB and FILE1 Table in RG22.

ASSEMBLE 4 I/O PHASE (RG28)

Chart: ED

Functions: Puts out object code for all indexed-sequential disk files.

Entry: RPG, from RG24, or from RG26.

Input: None.

Output: Object code in DSF to perform I/O for all indexed-sequential disk files.

External References: See Table 4.

Exit:

- Normal: To RG32 (Assemble Tables phase) or RG34 (Assemble Chain and Record Address Files phase).
- Error: None.

Tables/Work Areas: Tables built by this phase are all internal and describe the object code for the Disk System Format routine, ZPTOB.

ASSEMBLE TABLES PHASE (RG32)

Chart: FA

Functions:

- Checks compression for Table files.
- Builds Table file area.
- Generates table loading routine.
- Generates table dump routine.
- Generates linkage routine if more than one table loading or dumping routine has been generated.

Entry: A0000, from RG28.

Input: Compression Records built from Extension Specifications.

Output:

- Area (Save) for table entries.
- Object code for table load routine (at LD).
- Object code for table dump routine (at DP).
- Object code for table link routine (at LK).

External References: See Table 4

Exit:

- Normal: To RG34, Assemble Chain and RA File Phase.
- Error: None.

Tables/Work Areas: None.

ASSEMBLE CHAIN AND RA FILE PHASE (RG34)

Chart: FB

Functions:

- Processes Extension Compression.
- Generates object code for record address (RA) files, and chaining files.

Entry: BEGC1, from the calling phase.

Input: Extension compression.

Output:

- Object code for processing RA Files.
- Object code for C1, C2, or C3 Chaining Files.

External References: See Table 4.

Exits:

- Normal: To RG36 (Assemble Input Fields Phase).
- Error: None.

Tables/Work Areas: IOTAB and FILE1 Table (both created by RG22) are referenced.

ASSEMBLE INPUT FIELDS (RG36)

Chart: FC

Functions: Assembles Input Specifications (Field type).

Entry: BEG, from RG34.

Input: Input Specifications compression.

Output:

- Object code to move fields.
- Object code to test Field Record Relation Indicators and set on Resulting indicators.
- Linkage to Sterling Input Specifications.
- Object code for chaining fields.

External References: See Table 4.

Exits:

- Normal: RG38 (Assemble Control Levels phase).
- Error: None.

Tables/Work Areas: None.

ASSEMBLE CONTROL LEVELS PHASE (RG38)

Chart: FD

Functions:

- Generates object code for both field type and record type Input Specifications.
- Generates sequence check routine (NUSEQ), if numeric record type is present.
- If control levels are present, generates the object code which processes them.

Entry: BEG38, from RG36.

Input: I type and D type Input compression.

Output:

- Object code for processing control levels.
- Object code for determining record type.
- Object code for checking numeric sequence.

External References: See Table 4.

Exit:

- Normal: To RG40 (Assemble Multi-Files Phase).
- Error: None.

Tables/Work Areas: EXSP - Work area for processing compression.

ASSEMBLE MULTI-FILES PHASE (RG40)

Chart: FE

Functions: Generates routines that move matching fields from input area to matching field hold areas, and routines that compare matching fields to determine sequence of processing and status of MR indicator.

Entry: BEG49, from RG38.

Input: I type and D type Input compression.

Output: Object code for processing matching fields.

External References: See Table 4.

Exits:

- Normal: RG42 (Assemble Get Phase).
- Error: None.

Tables/Work Areas: FILE1 Table (in RG22) is modified.

ASSEMBLE GET PHASE (RG42)

Chart: FF

Functions: Builds table (FILTA) containing addresses of input record routine (INPR), control level routine, move fields routine (INPF), and resulting indicators.

Entry: BEG40, from RG40.

Input: I type and D type Input compression.

Output:

- Object time file processing table for each file.
- A Get routine for each primary and secondary file.

External References: See Table 4.

Exits:

- Normal: To RG44, RG66, or RG52.
- Error: None.

Tables/Work Areas:

- FILE1 Table is modified.
- FILTA, work area for building object time file tables is built.

ASSEMBLE CALCULATION 1 PHASE (RG44)

Chart: FG

Functions:

- Assemble the Object Code routine for each LOKUP operation.
- Assembles a chain subroutine which may be linked to by any CHAIN operation.

Entry: BETG, from RG42.

Input: Calculation compression.

Output:

- Object code for CHAIN and LOKUP operations.
- Address of chain subroutine placed in compression for each chain operation.
- Address of each LOKUP routine placed in corresponding LOKUP compression.

External References: See Table 4.

Exits:

- Normal: To RG46, when all Calculation Specifications have been processed.
- Error: None.

Tables/Work Areas: None.

ASSEMBLE CALCULATION 2 PHASE (RG46)

Chart: FH

Functions:

- Generates object code for all Calculation Specifications except CHAIN and LOKUP.
- Generates linkage to CHAIN and LOKUP routines assembled in RG44.

Entry: BEG, from RG44.

Input: Calculation Specification compression.

Output: Object code for all Calculation Specifications.

External References: See Table 4.

Exit:

- Normal: To RG52 (Assemble Output Fields).
- Error: None.

Tables/Work Areas: OPERA - A table of operation codes is built.

ASSEMBLE OUTPUT FIELDS (RG52)

Chart: FI

Functions: Generates object code that will place output fields in desired format and location within the associated output record.

Entry: RPG, from RG46.

Input: Output Compression.

Output: Object code.

External References: See Table 4.

Exit:

- Normal: RG54 (Assemble Put phase).
- Error: None.

Tables/Work Areas: None.

ASSEMBLE PUT PHASE (RG54)

Chart: FJ

Functions:

- Generates object code that produces output records on output files.
- Fills in NOTES, WORK1, SQSOF, and TABL1.

Entry: RPG, from RG52.

Input: Output Compression.

Output: Object code and a table of addresses.

External References: See Table 4.

Exit:

- Normal: RG58, (Assemble Linkage Phase).
- Error: None.

Tables/Work Areas:

- NOTES - Work area in COMAREA.
- WORK1 - A 3-word area for building one table entry.
- SQSOF - Overflow table (see CONTROL BLOCKS AND TABLES).
- TABL1 - Area where Output tables are built.

ASSEMBLE LINKAGE PHASE (RG58)

Chart: FK

Functions:

- Generates a fixed driver (at R0) for object time execution.
- Generates branches to appropriate routines if OPENs and CLOSEs are needed.
- Generates a link to table load, if needed.
- Generates link to Heading and Detail Lines routine.

Entry: BEGIN, from RG54.

Input: None.

Output:

- A printed listing of key addresses of object program.
- A printed listing of the number of sectors needed for ISAM LOAD files (providing list option and ISAM LOAD are specified).
- Object code as described under "Function", above.

External References: See Table 4.

Exits:

- Normal: To RG60.
- Error: None.

Tables/Work Areas:

- ISAM Load Table - contains information concerning sector count.
- Filename Table - contains names of specified files (see "CONTROL BLOCKS AND TABLES").
- FILE1 Table - contains information about file types (see "CONTROL BLOCKS AND TABLES").

TERMINATE COMPILATION (RG60)

Chart: FL

Functions:

- Updates DCOM (Disk Communications Region) on the system and working storage cartridges for the DSF program, and moves the DSF program to the beginning of working storage if there are no terminating errors.
- Sets words in system area so DUP and XEQ cards will be passed if there are any terminating errors.
- Prints "end of compilation" message and passes control to the Monitor.
- Calls OBEND (wrap-up routine in RG10).

Entry: BEGIN, from RG58.

Input:

- DCOM from system and working storage cartridges.
- DSF program (if it does not start at the beginning of working storage).

Output:

- Updated DCOM.
- DSF program moved to the beginning of working storage.
- If any terminal errors, DUP and XEQ are disabled.

External References: See Table 4.

Exit:

- Normal: To the Monitor, EXIT.
- Error: To the Monitor (when $NDUP and &NXEQ are non-zero), via EXIT.

Tables/Work Areas: DISK - Work area for reading from the disk.

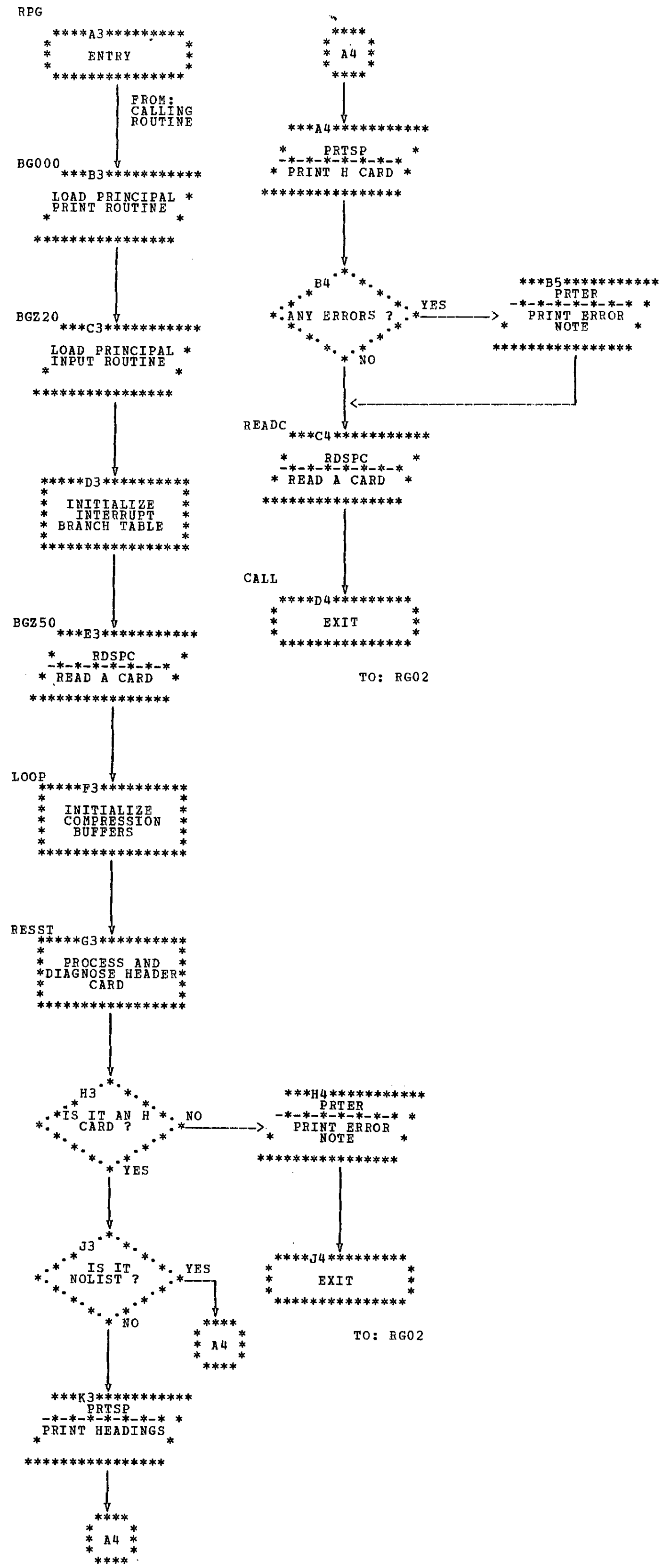


Chart AA. Resident Phase (RG00)

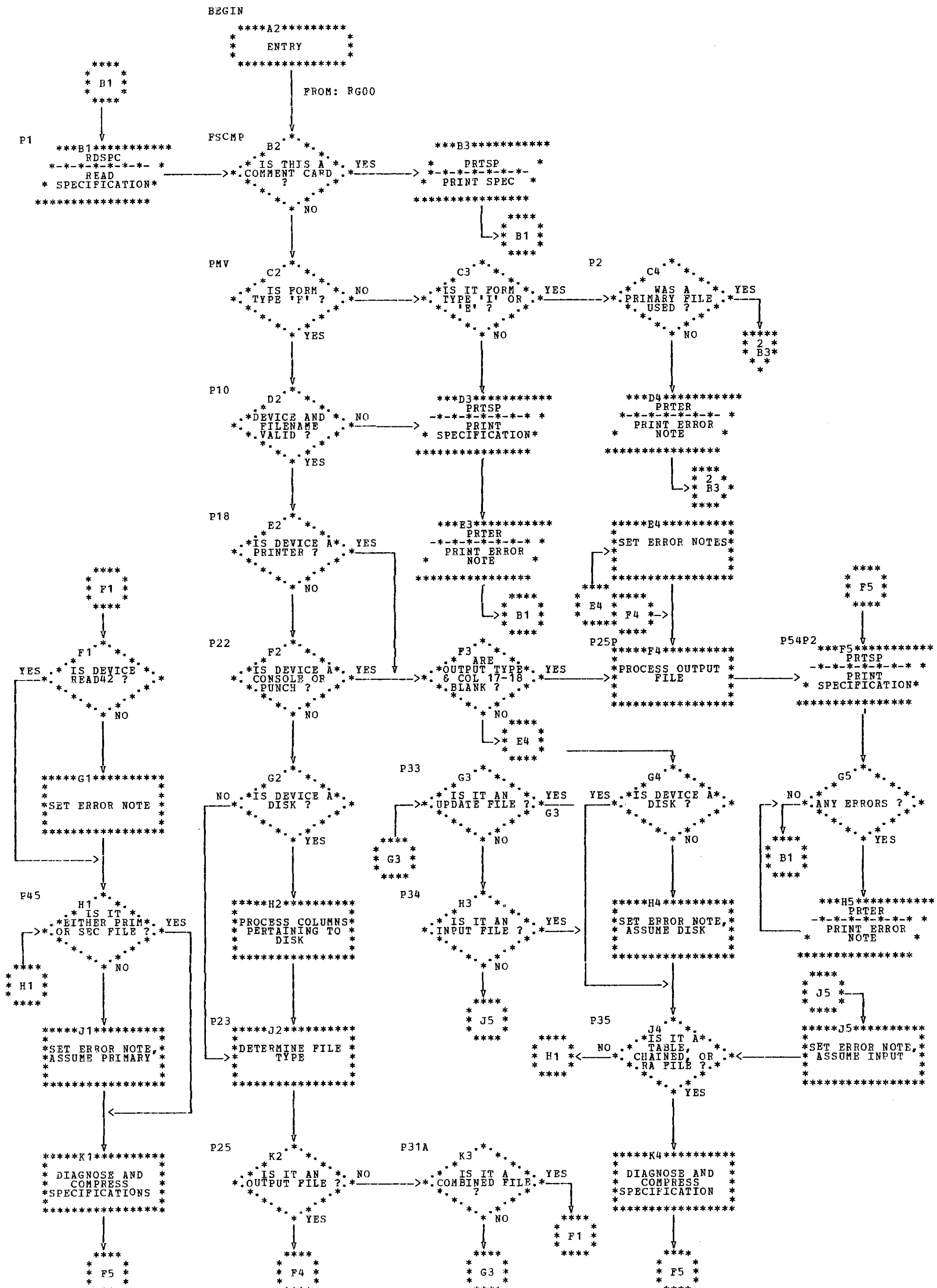


Chart BA. Enter File Specifications Phase (RG02)

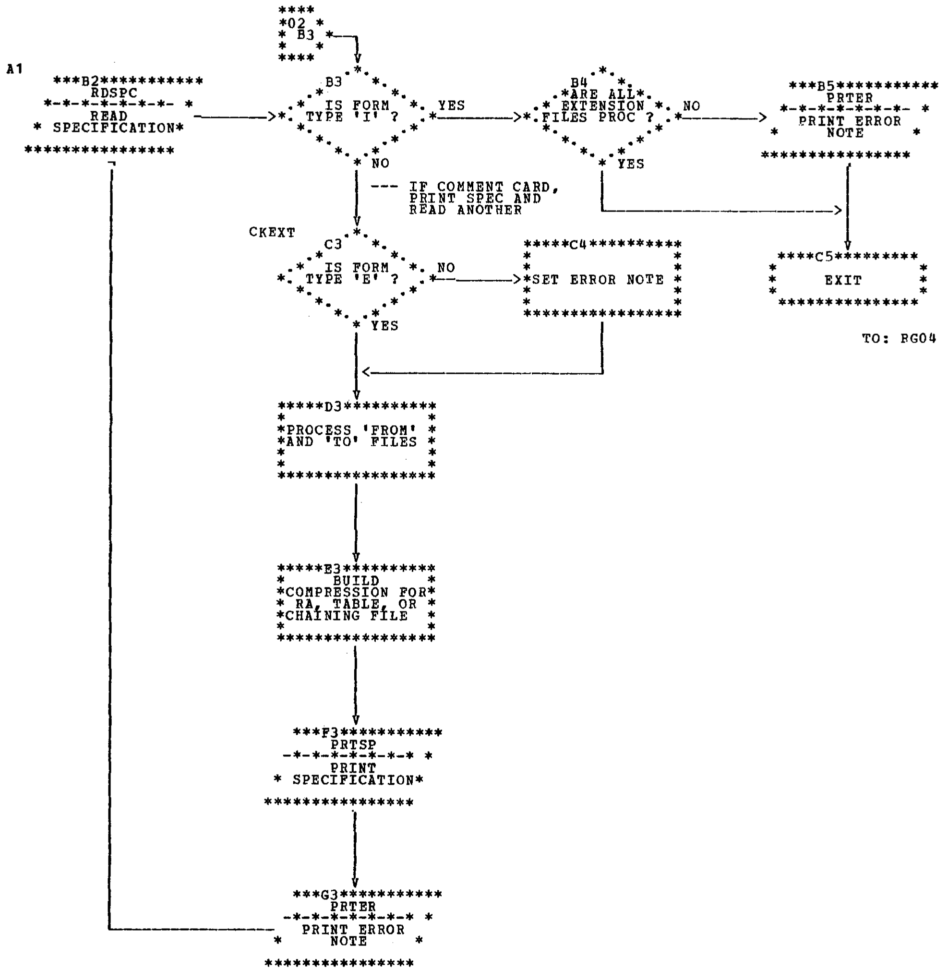


Chart BA. Enter File Specifications Phase (RG02)

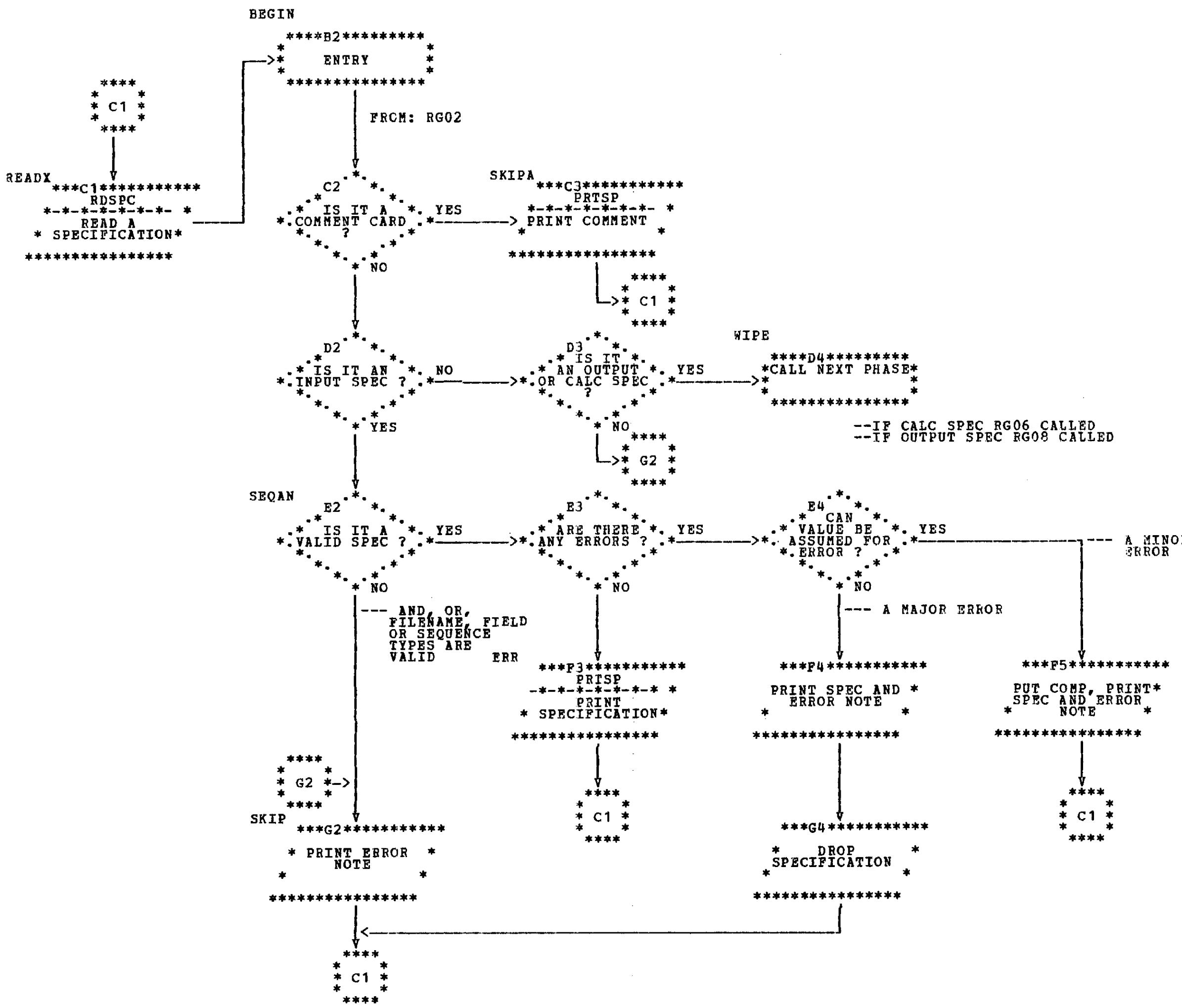


Chart BB. Enter Input Specifications Phase (RG04)

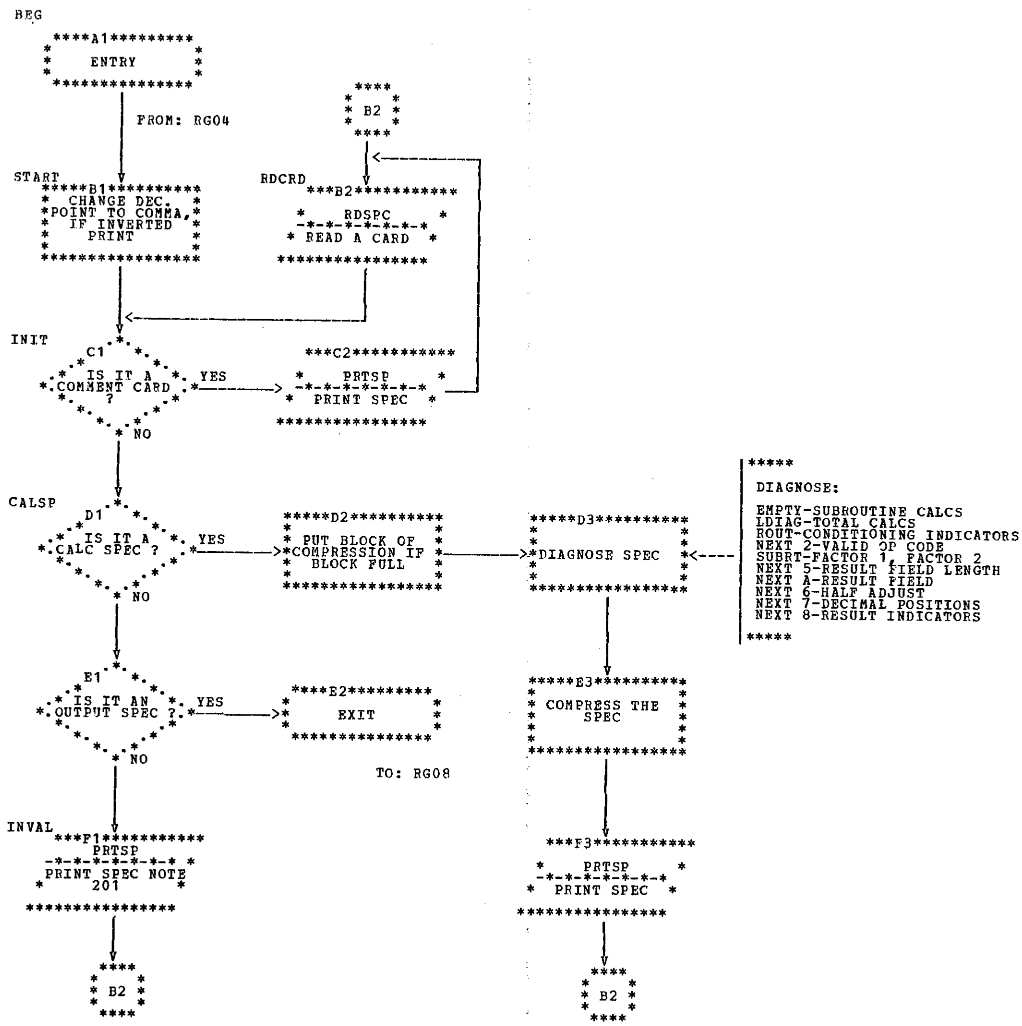


Chart BC. Enter Calculations Specifications Phase (RG06)

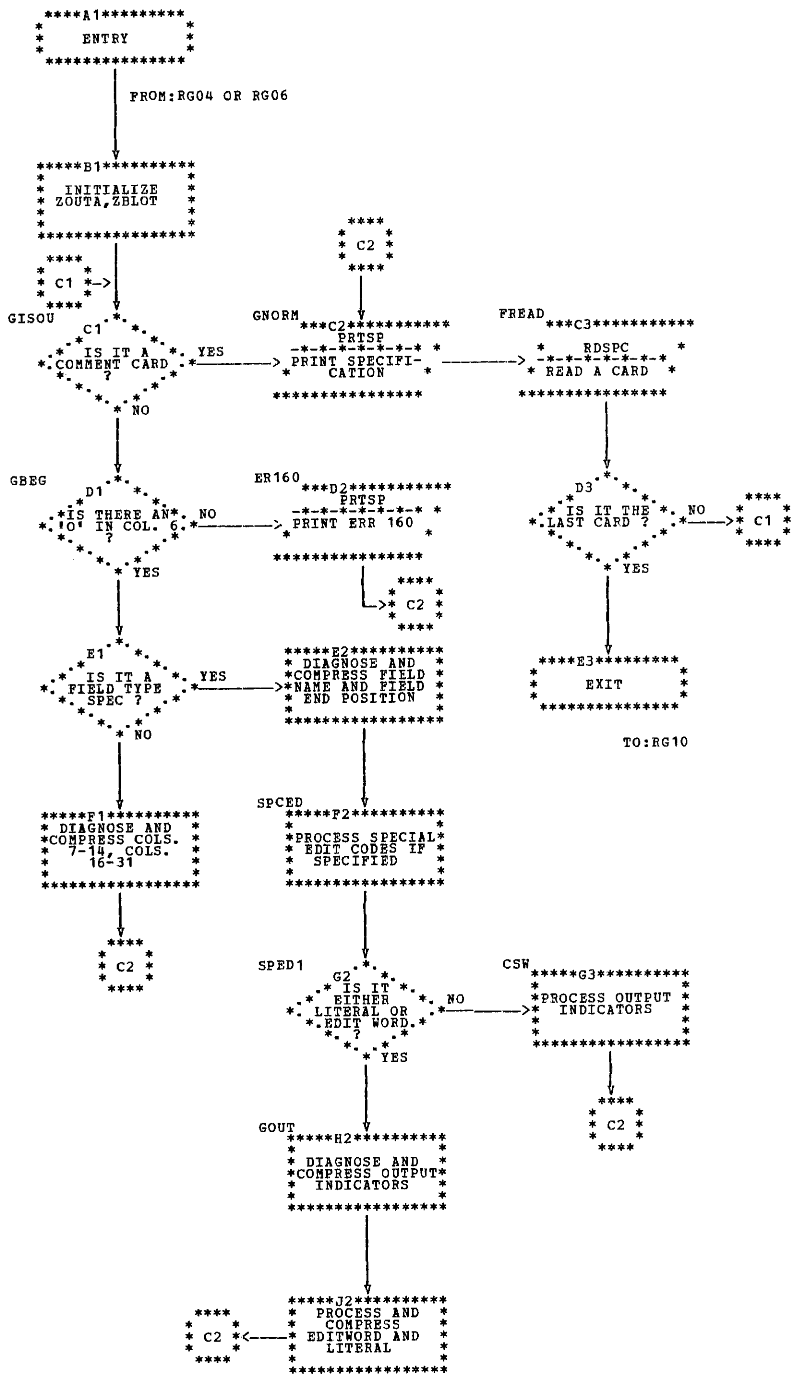


Chart BD. Enter Output-Format Specifications Phase (RG08)

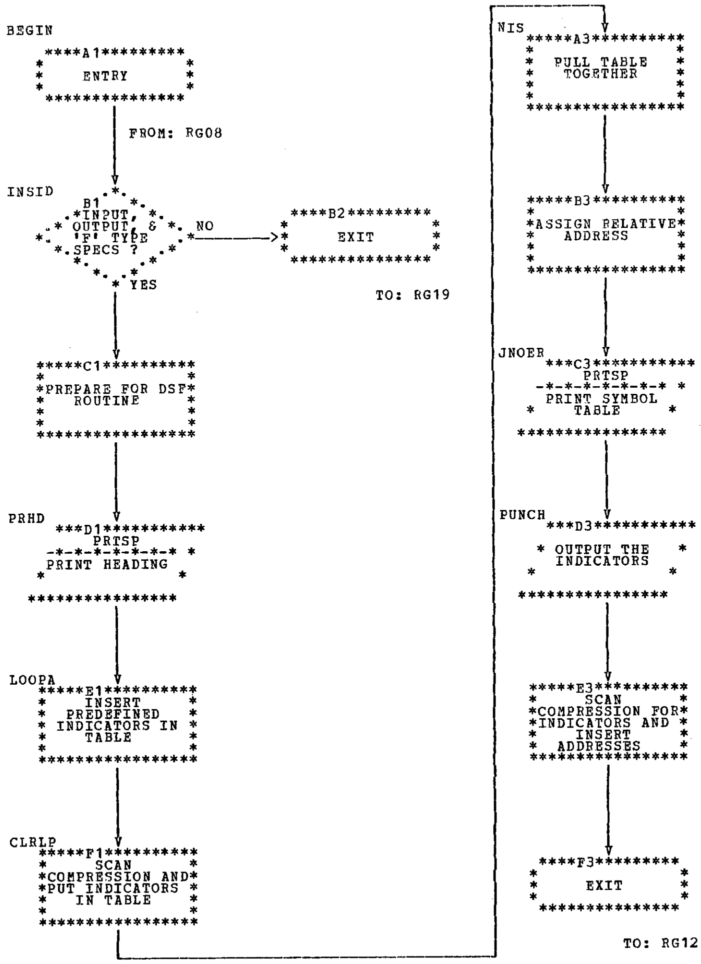


Chart CA. Assign Indicators Phase (RG10)

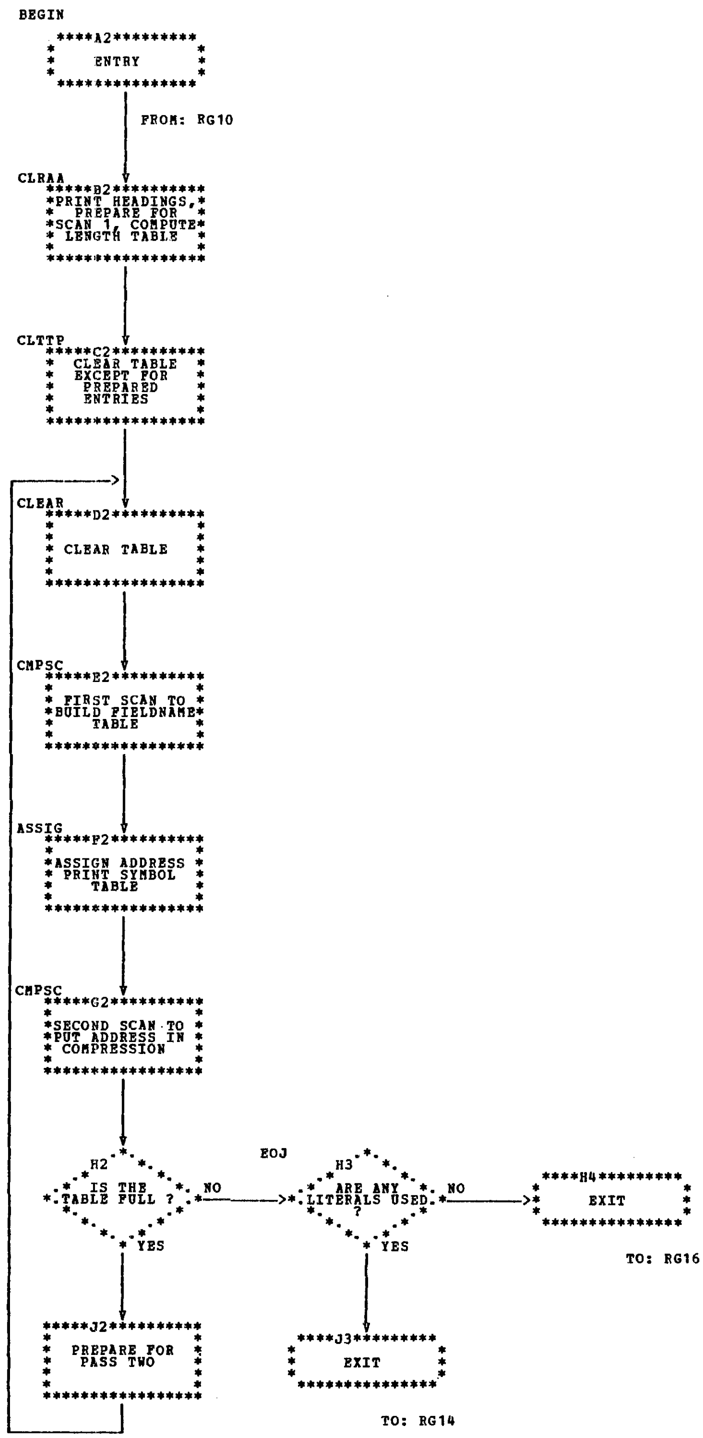


Chart CB. Assign Field Names Phase (RG12)

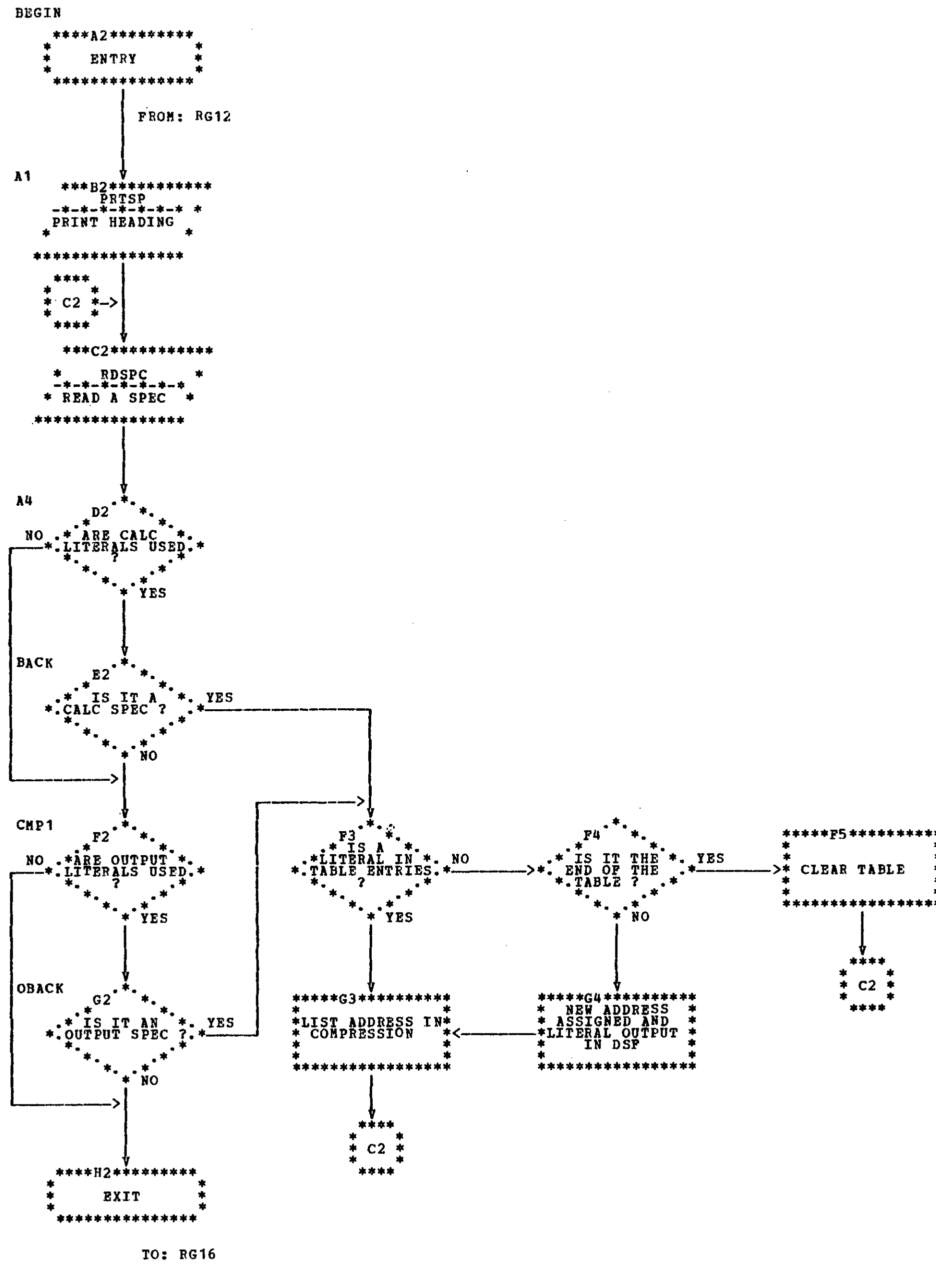
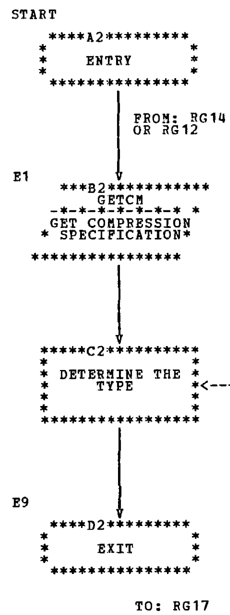


Chart CC. Assign Literals Phase (RG14)



ADDRESS	TYPE	ACTION TAKEN
E1	F	AT E2, BUILD TENT TABLE, BRANCH TO E1
--	R OR E	BRANCH TO E1
--	T	CHECK FOR VALID TABLE NAME, BRANCH TO E1
E1A	I	CHECK FOR 'AND' RECORD AT E6A; IF PRESENT, BRANCH TO E6E, IF NOT BRANCH TO E1A
--	D	AT E7, CHECK COLUMNS 42-74 FOR VALIDITY, BRANCH TO E1
--	C	TEST VALIDITY OF INPUT INFROMATION, BRANCH TO E9
--	O	BRANCH TO E9

Chart DA. Extended Diagnostics Phase (RG16)

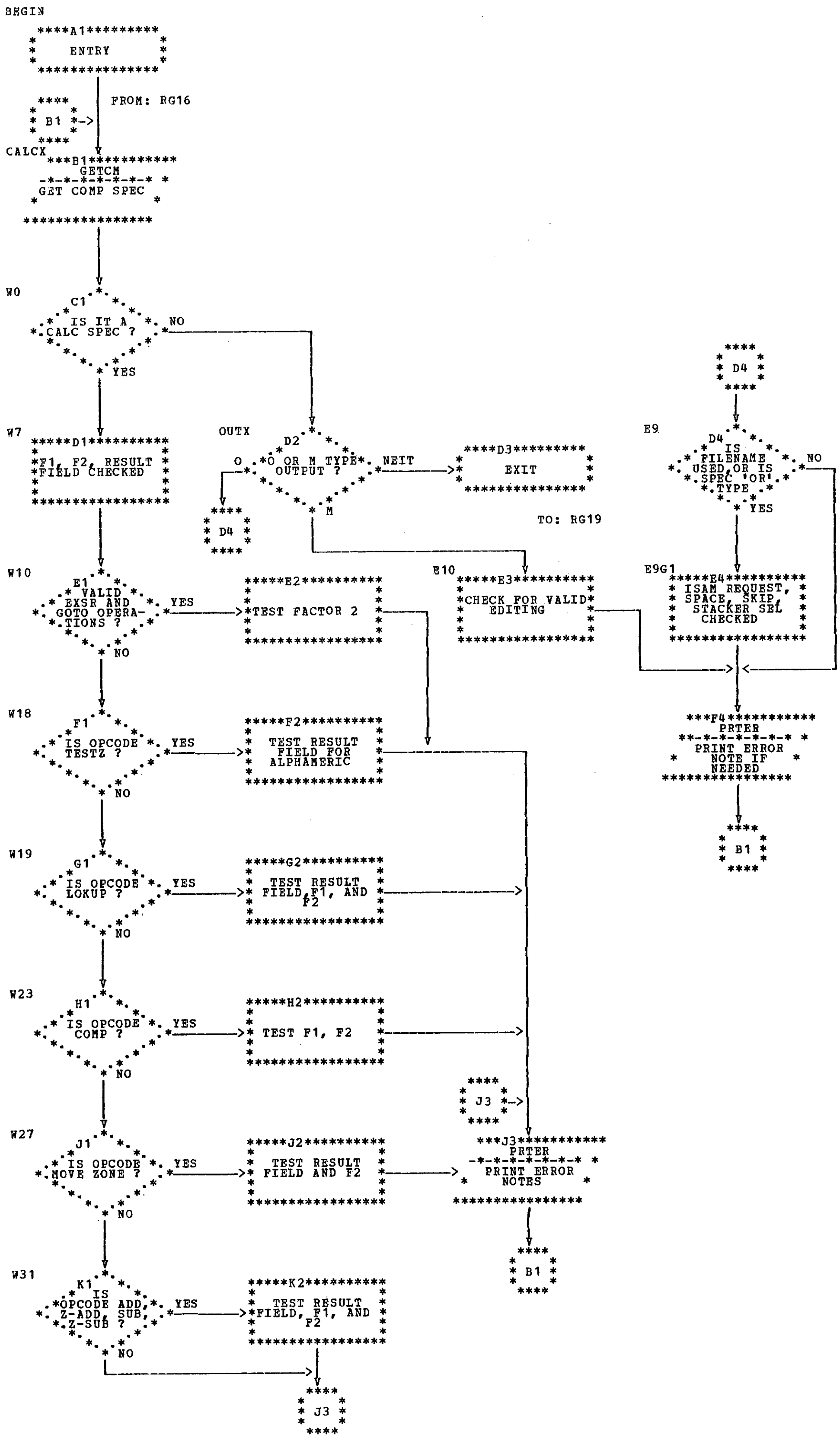


Chart DB. Extended Calculation and Output Diagnostic Phase (RG17)

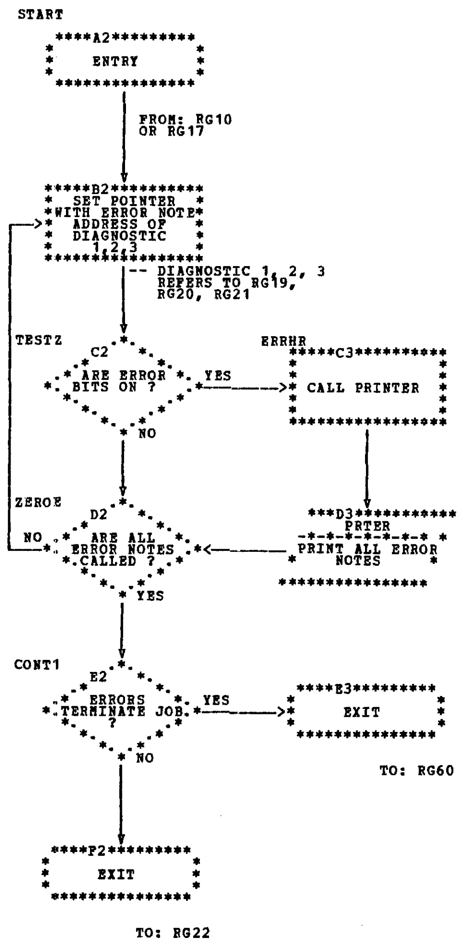


Chart DC. Error Message Phases (RG19, RG20, RG21)

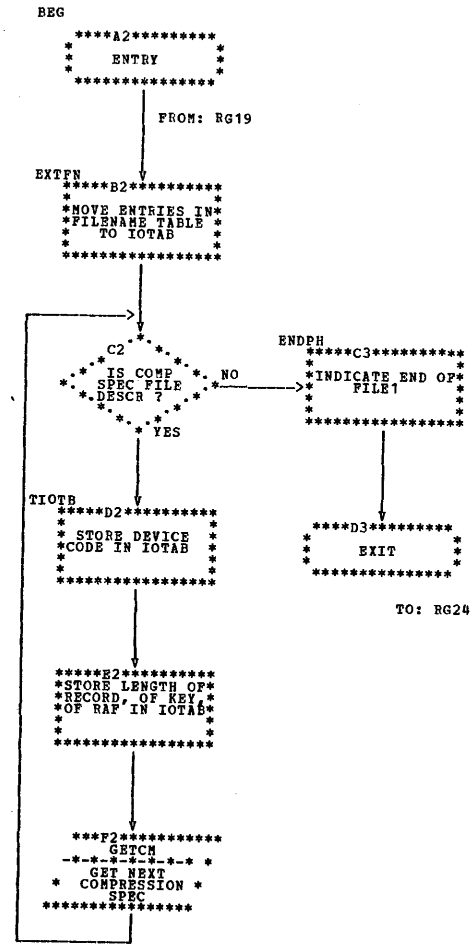


Chart EA. Assemble 1 I/O Phase (RG22)

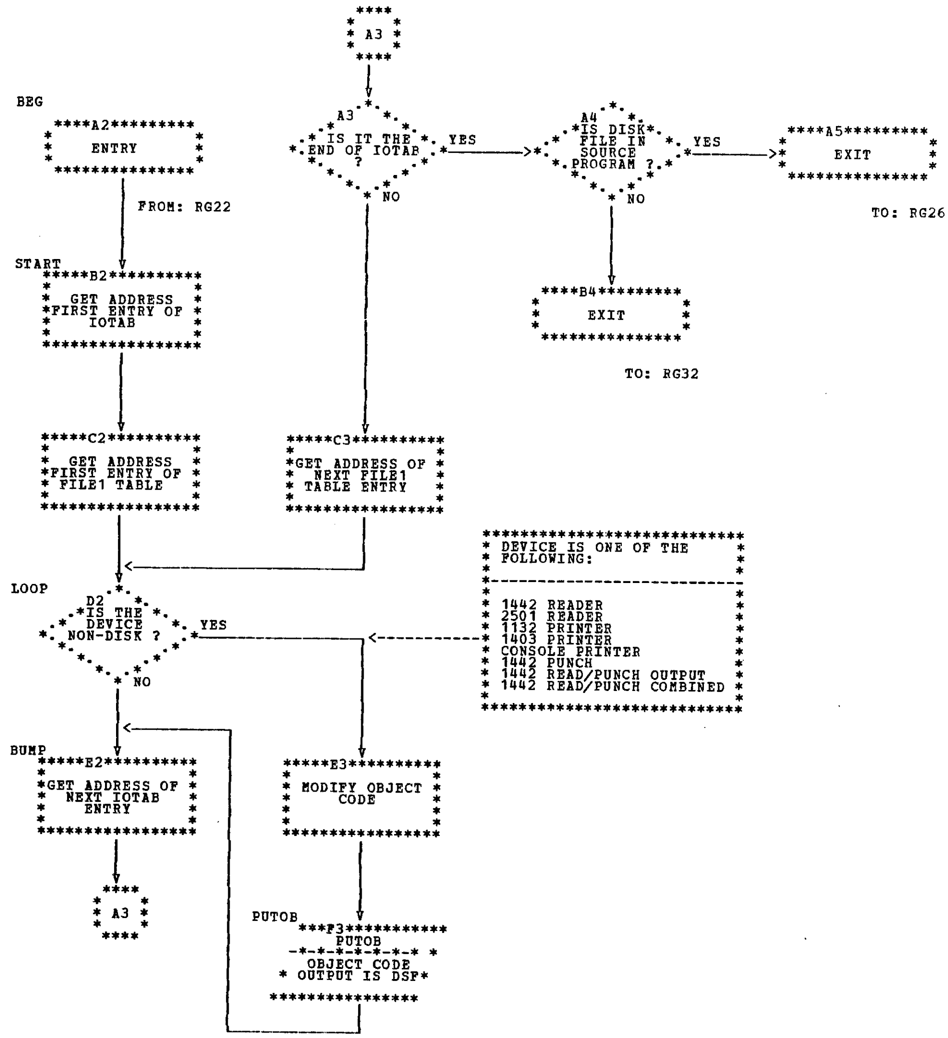


Chart EB. Assemble 2 I/O Phase (RG24)

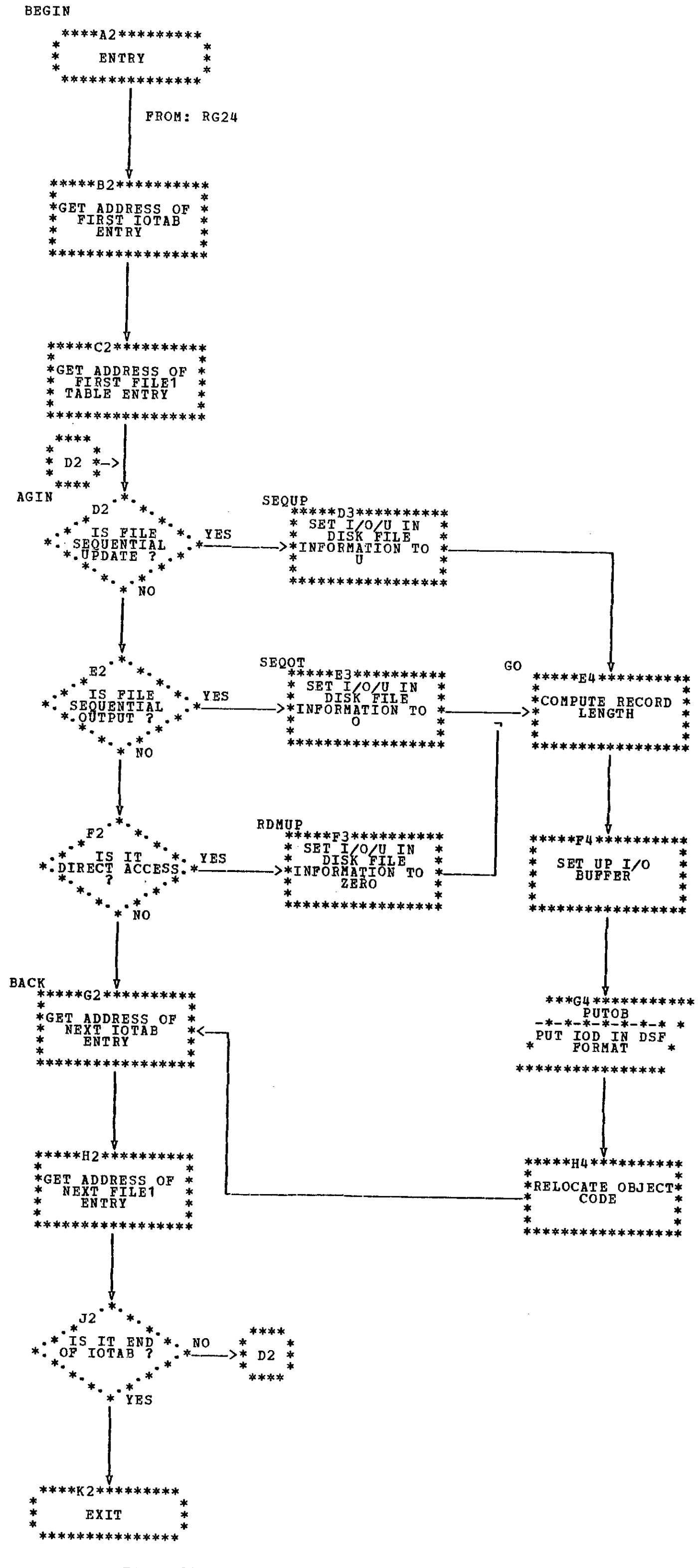


Chart EC. Assemble 3 I/O Phase (RG26)

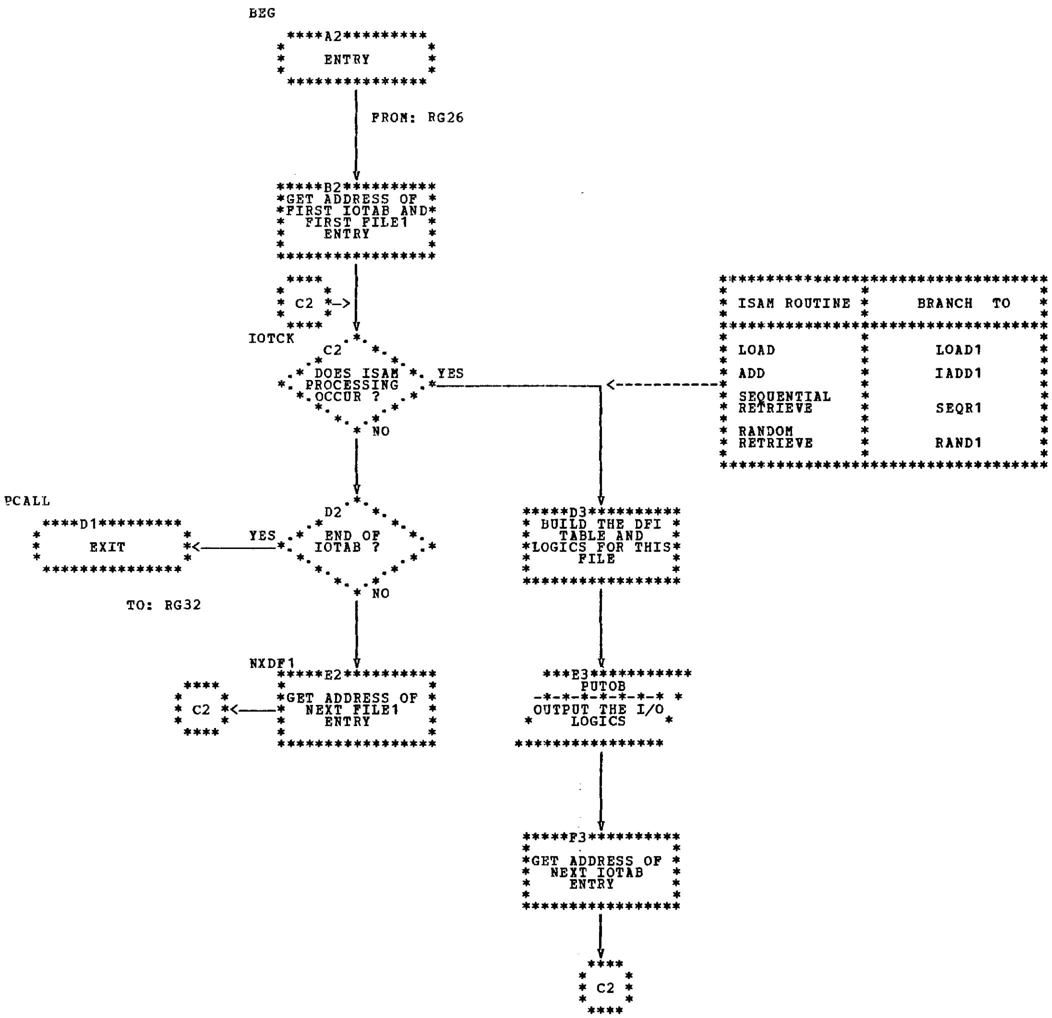


Chart ED. Assemble 4 I/O Phase (RG28)

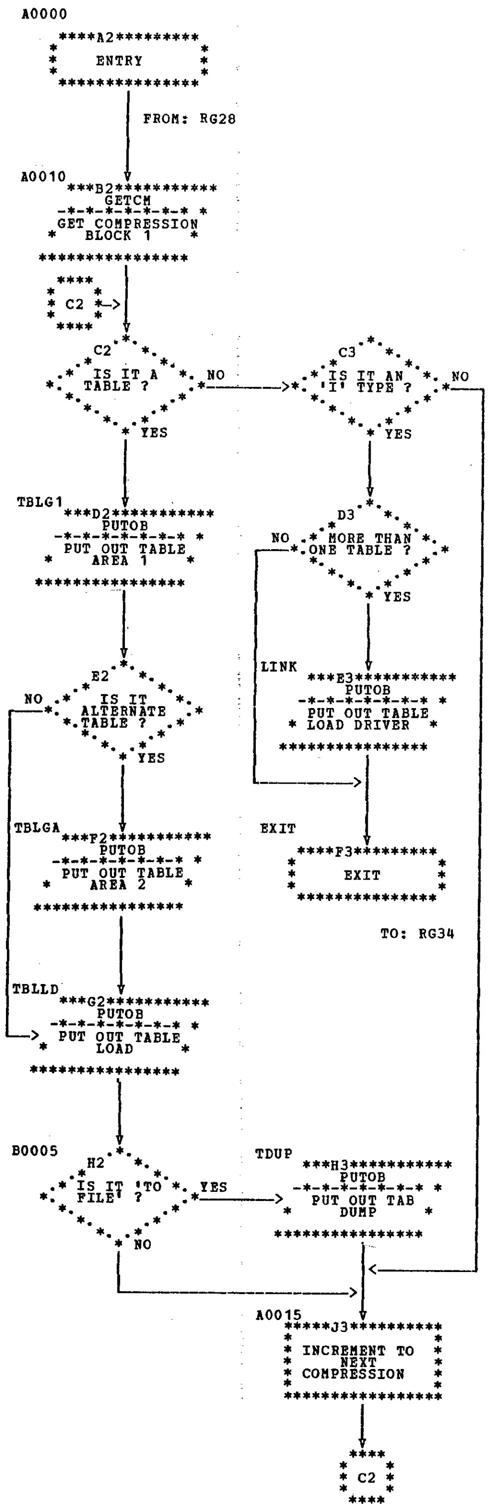


Chart FA. Assemble Tables Phase (RG32)

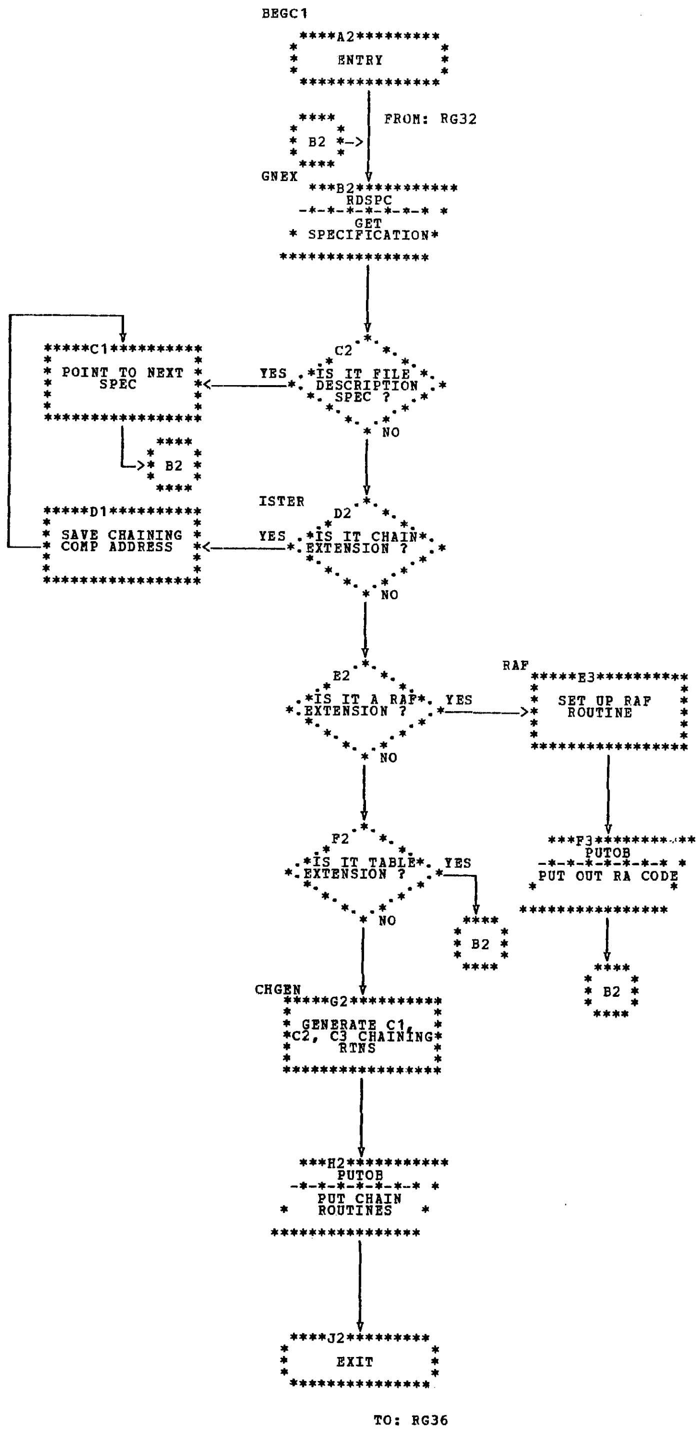


Chart FB. Assemble Chain and RA File Phase (RG34)

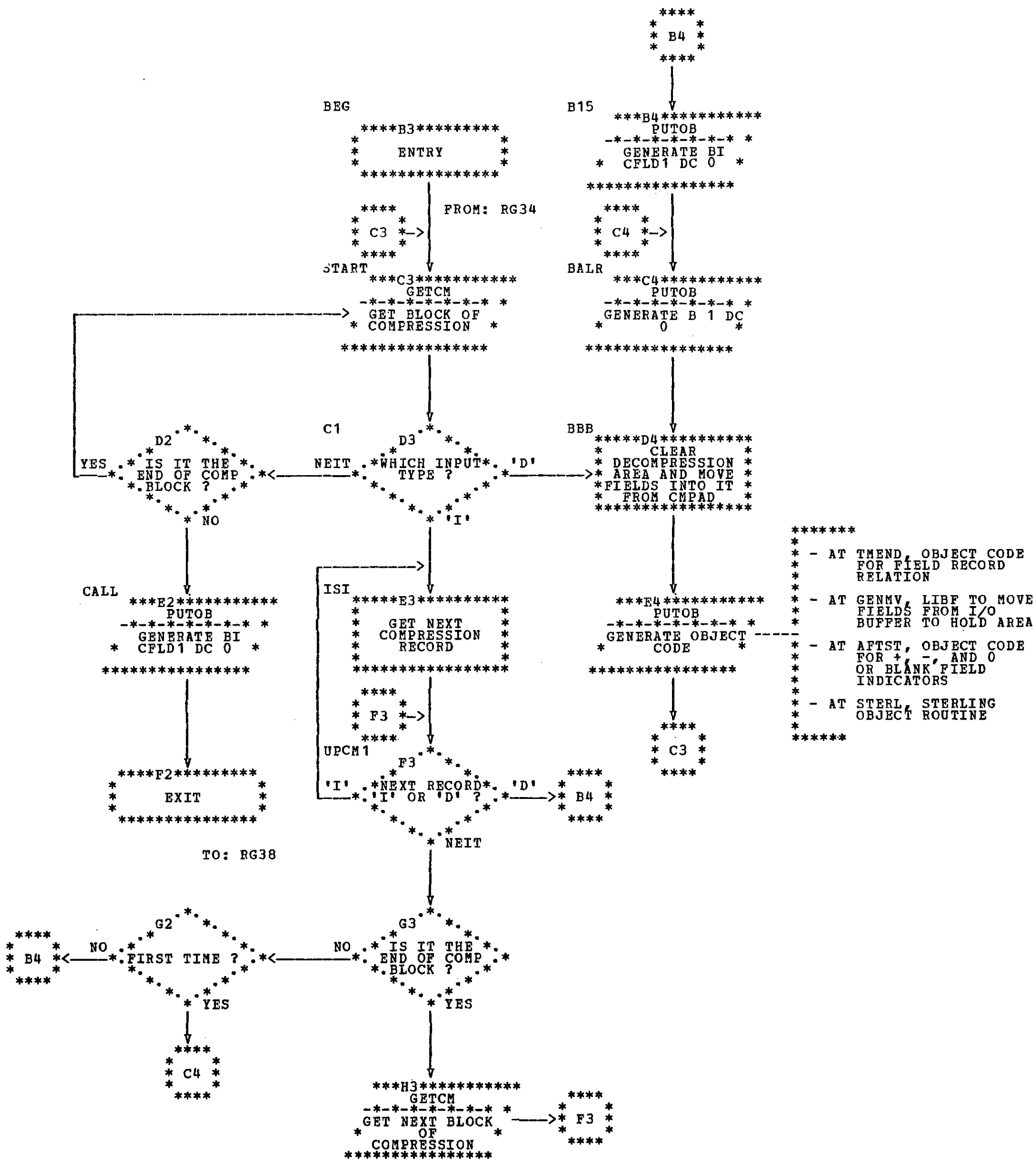


Chart FC. Assemble Input Field Phase (RG36)

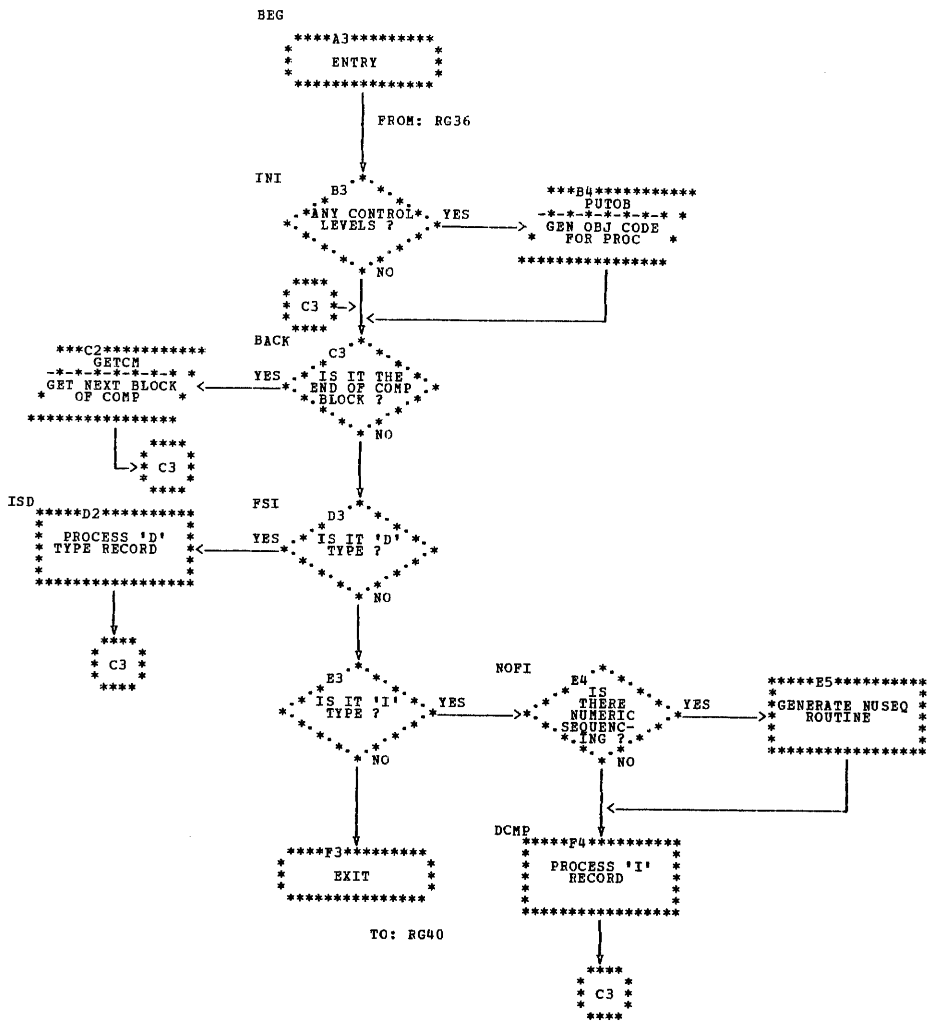


Chart FD. Assemble Control Levels Phase (RG38)

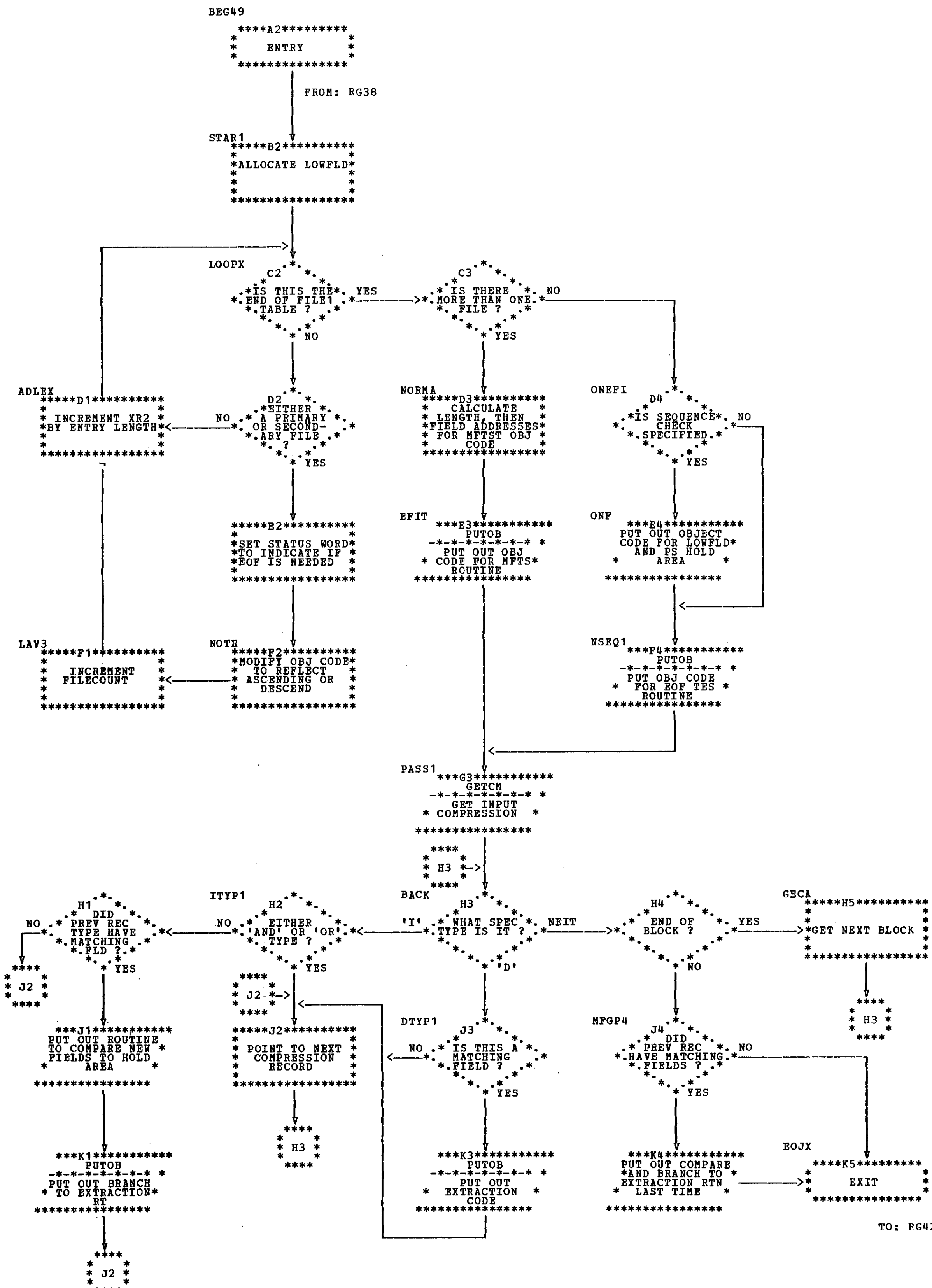


Chart FE. Assemble Multi-Files Phase (RG40)

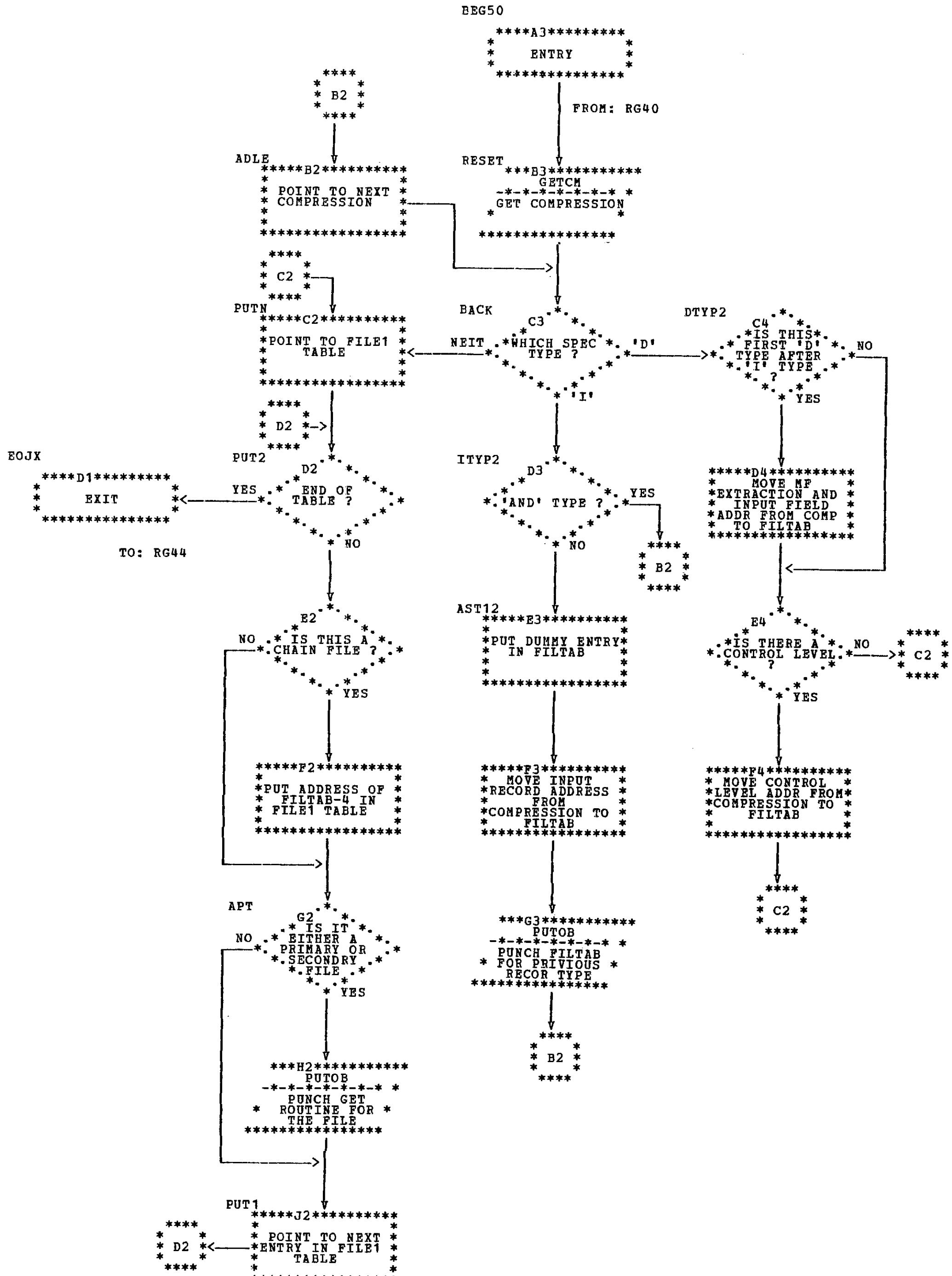
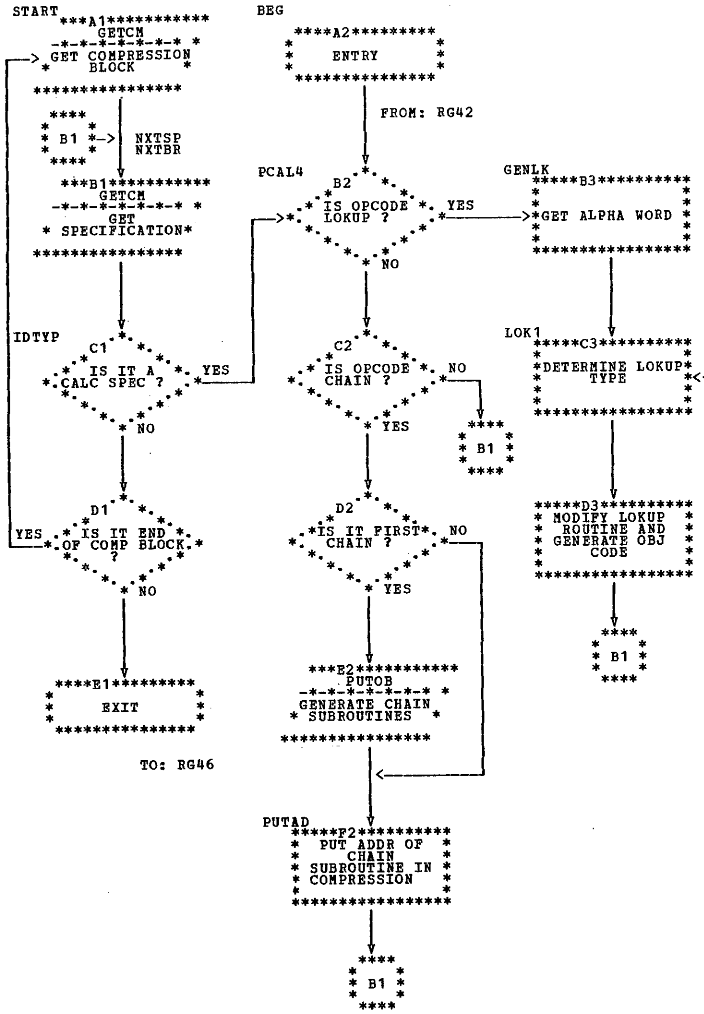


Chart FF. Assemble GET Phase (RG42)



LOKUP TYPE	ARGUMENT TABLE ASCENDING OR DESCENDING	LOKUP INSTRUCTION TYPE IN WHICH ADDRESS IS STORED
EQUAL	A OR D	I
HIGH	A	I
	D	III
LOW	A	III
	D	I
HIGH, EQUAL	A	II
	D	IV
LOW, EQUAL	A	IV
	D	II

Chart FG. Assemble Calculation 1 Phase (RG44)

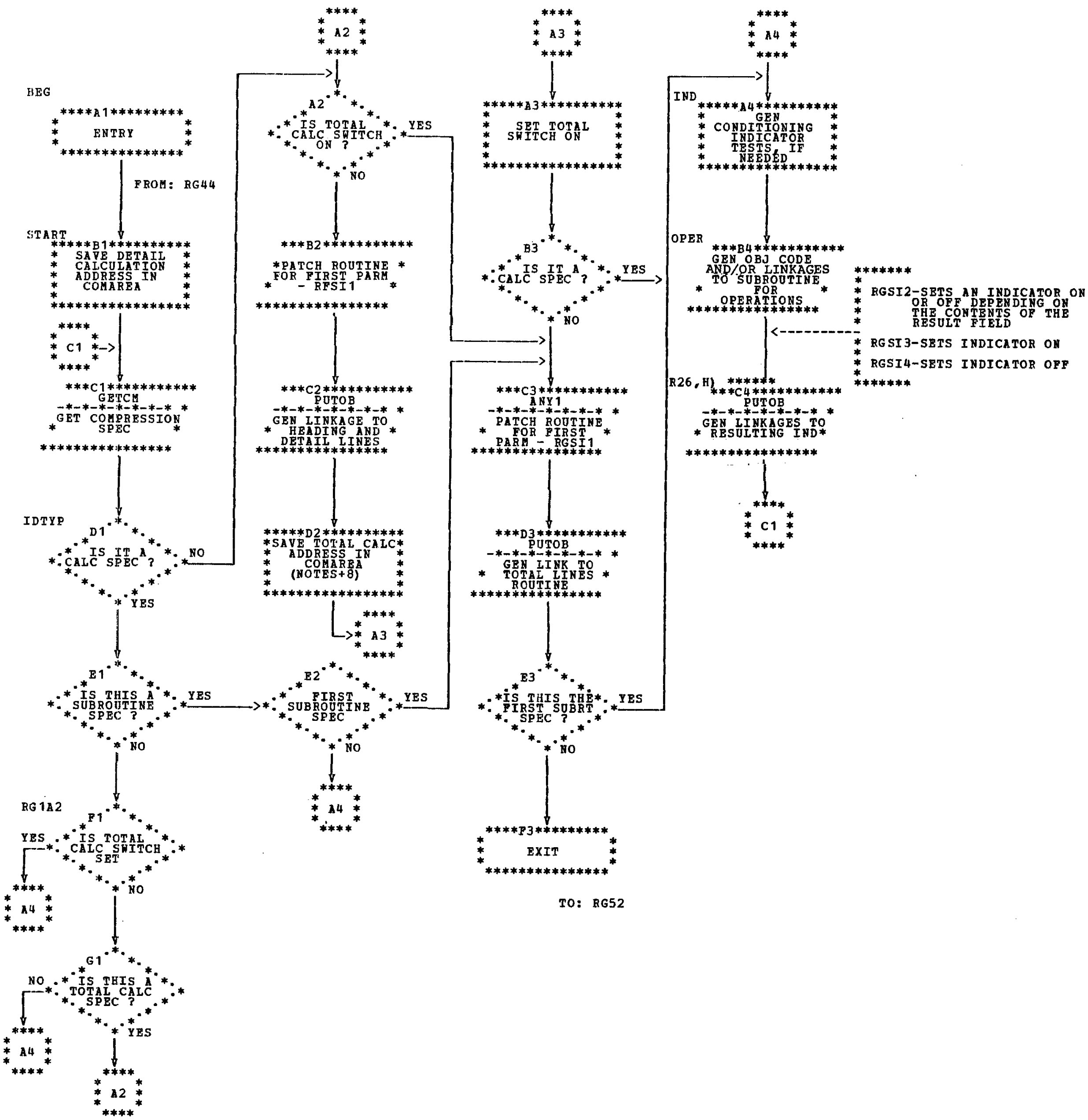


Chart FH. Assemble Calculation 2 Phase (RG46)

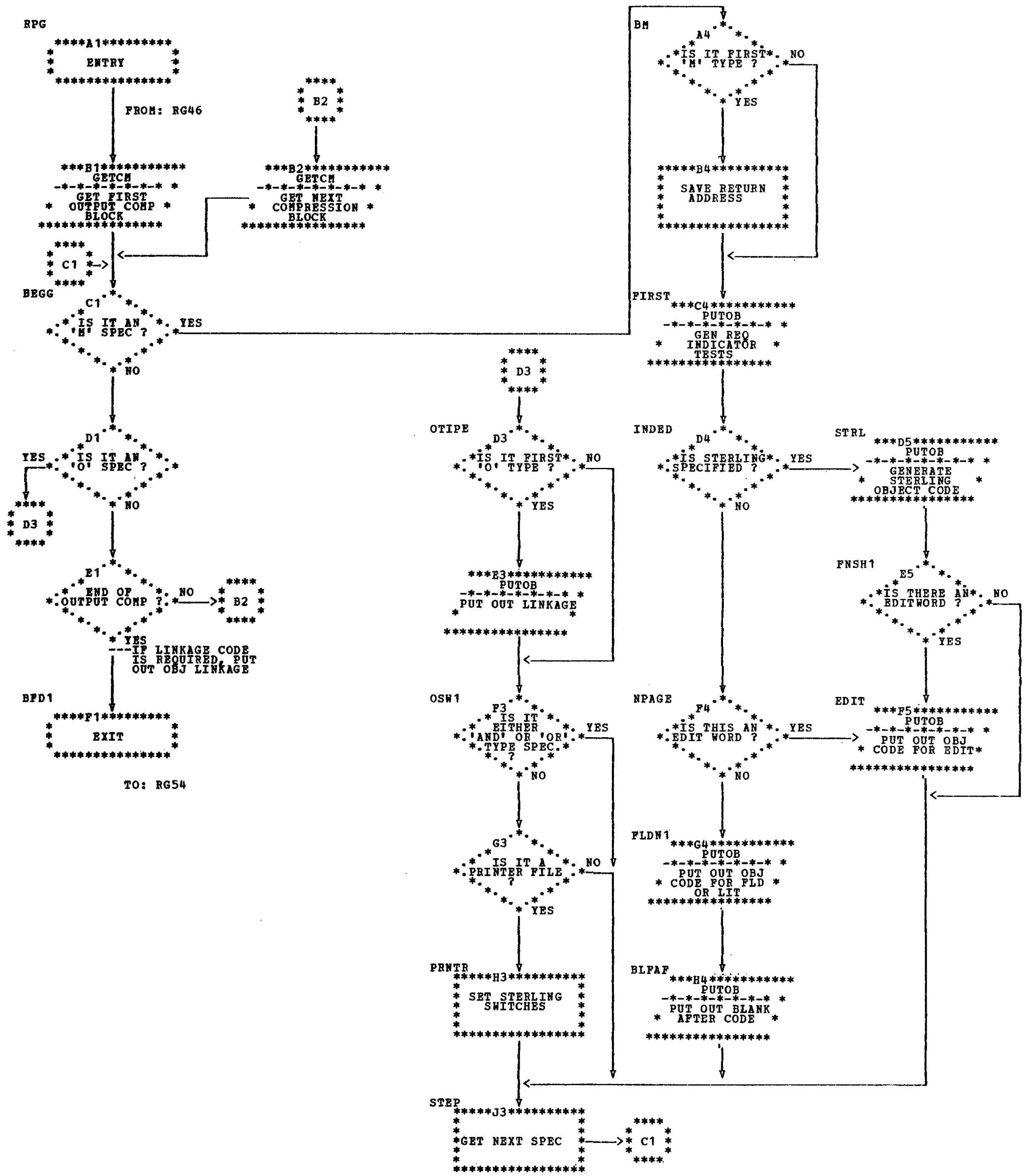


Chart FI. Assemble Output Fields Phase (RG52)

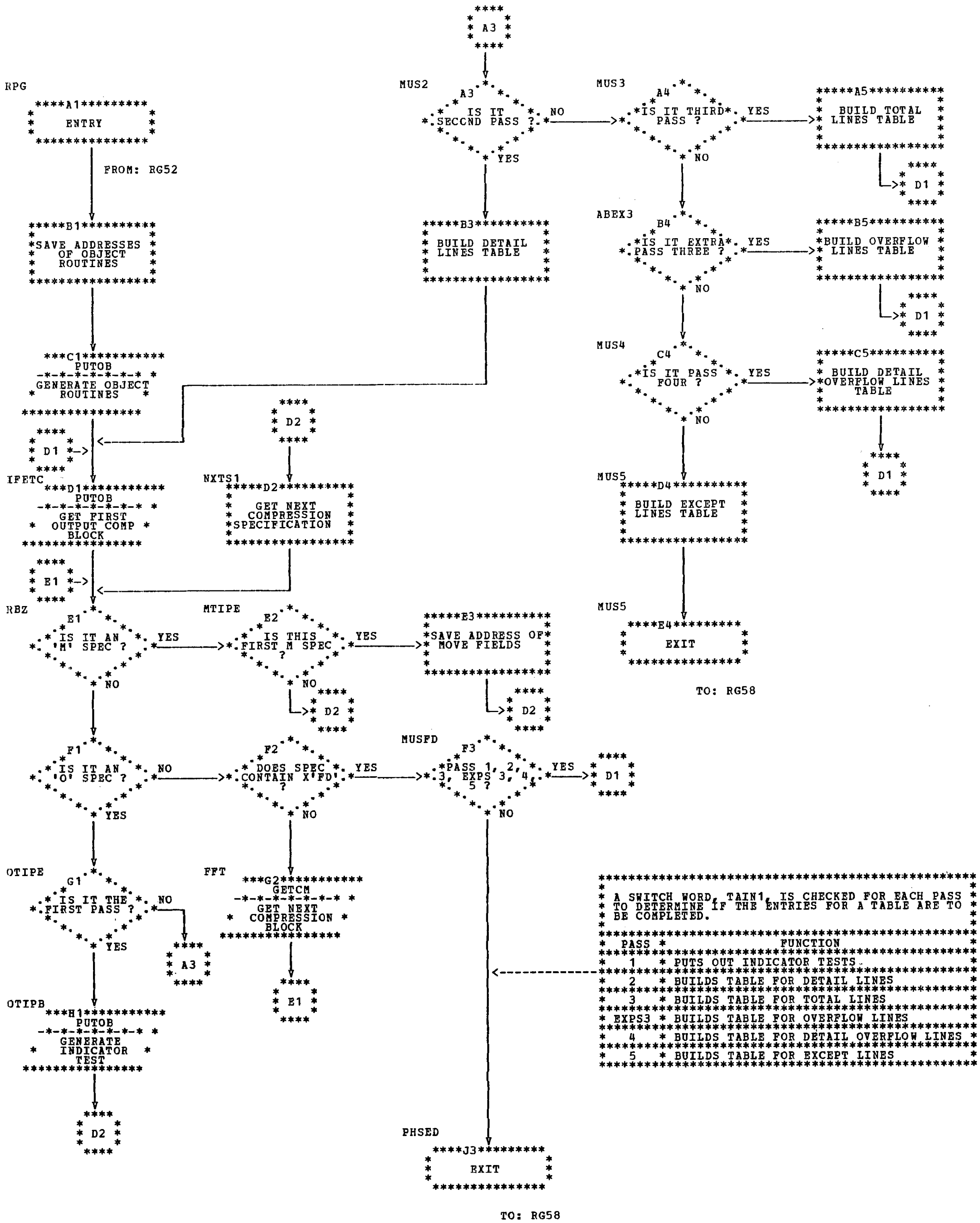


Chart FJ. Assemble PUT Phase (RG54)

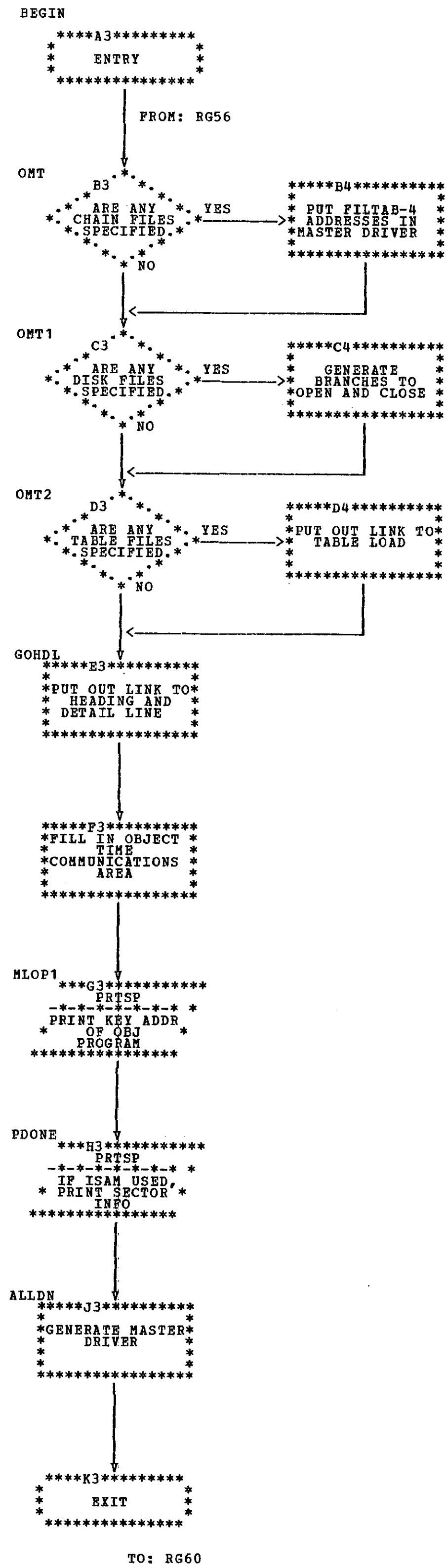


Chart FK. Assemble Linkage Phase (RG58)

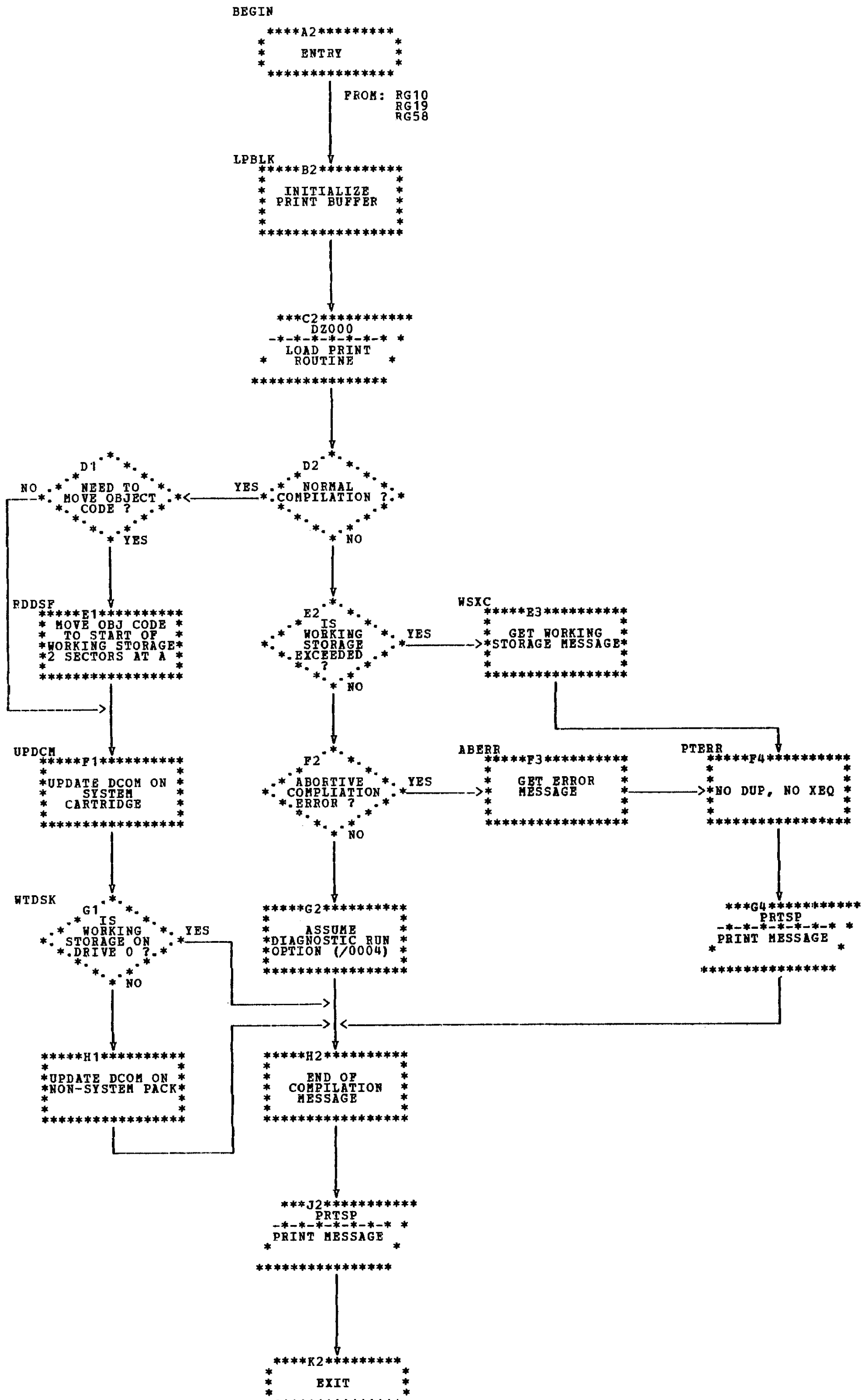


Chart FL. Terminate Compilation Phase (RG60)

The Phase Directory lists the 29 phases of the 1130 RPG Compiler in numeric sequence. It summarizes the operations of each phase, and lists the corresponding Module Name and Point of Entry. Chart ID identifies the appropriate flowchart for each phase. (See Compiler Flowcharts.)

Module Name	Generic Name	Chart ID	Entry Point	Synopsis of Functions
RG00	Resident Phase	AA	RPG	Load subroutines, provide a commonly accessed communication area, read RPG control card, and print headings.
RG02	Enter File Specifications Phase	BA	BEGIN	List and compress File Specification entries, build compression areas and Filename Table.
RG04	Enter Input Specifications Phase	BB	BEGIN	Process Input Specifications creating compression records, check I and D specification, and analyze errors.
RG06	Enter Calculation Specifications	BC	BEG	Read, list, and compress Calculation Specifications.
RG08	Enter Output-Format Specifications Phase	BD	RPG	Read, list, and compress Output-Format Specifications and determine whether it defines a record type or a field type.
RG10	Assign Indicators Phase	CA	BEGIN	Build an indicator table, replace the indicators with addresses, place PUTOB in RG00, create OBEND and place it in RG00.
RG12	Assign Field Names Phase	CB	BEGIN	Build table of field names, and replace a field in compression with its corresponding address.
RG14	Assign Literals Phase	CC	BEGIN	Assign addresses to literals and build edit words from edit codes.
RG16	Extended Diagnostics 1 Phase	DA	START	Build tables (TENT, ERTAB, and NOTAB), distinguish record type from field type Input Specifications, and pass on field lengths to the Assemble phases.
RG17	Extended Diagnostics 2 Phase	DB	BEGIN	Update table address, check field entries, print error notes, put out control level hold areas.
RG19, 20, 21	Diagnostic Message Phases (1, 2, and 3)	DC	START	Diagnose error bits and print error messages.
RG22	Assemble 1 I/O Phase	EA	BEGIN	Build I/O Table from Filename Table.

Table 2. Phase Directory (Part 1 of 2)

Module Name	Generic Name	Chart ID	Entry Point	Synopsis of Functions
RG24	Assemble 2 I/O Phase	EB	BEG	Produce object code for I/O requests of non-disk files.
RG26	Assemble 3 I/O Phase	EC	BEGIN	Produce object code for I/O requests of sequential disk files and Direct access disk files.
RG28	Assemble 4 I/O Phase	ED	RPG	Put out object code for indexed-sequential disk files.
RG32	Assemble Table Phase	FA	A0000	Put out object code to load and dump tables.
RG34	Assemble Chain and RA File Phase	FB	BEGC1	Process compression and generate object code.
RG36	Assemble Input Fields Phase	FC	BEG	Assemble field type Input Specifications.
RG38	Assemble Control Levels Phase	FD	BEG38	Generate control level object code and sequence check routine.
RG40	Assemble Multi-Files Phase	FE	BEG49	Generate Matching Field routines.
RG42	Assemble Get Phase	FF	BEG50	Build file table of routine addresses.
RG44	Assemble Calculation 1 Phase	FG	BEG	Assemble LOKUP operations and a chain sub-routine.
RG46	Assemble Calculation 2 Phase	FH	BEG	Generate object code and linkage.
RG52	Assemble Output Fields	FI	RPG	Generate object code routine placing output fields in desired format.
RG54	Assemble Put Phase	FJ	RPG	Generate object code to put out records and produce table of addresses.
RG58	Assemble Linkage Phase	FK	BEGIN	Generate master driver, branches to routines, and linkage.
RG60	Terminate Compilation Phase	FL	RPG	Update WS cartridges, print end of compilation message.

Table 2. Phase Directory (Part 2 of 2)

This section describes tables and data areas in the RPG compiler that are used outside the phase which created them. (See Table 5, for more information.) Descriptions of the following are included:

- Filename Table
- TENT Table.
- Input-Output Table.
- Control Level Address Table.
- Overflow Table.
- FILE1 Table.
- Communication Area.

Filename Table

The Filename Table can contain as many as ten entries, each of which is four words long. If the number of entries exceeds 10 (overflows), the Enter File Specification phase (RG02), which builds this table, treats the additional entries as comments and prints an error.

Before an entry is placed in the table, the table is searched to determine if the entry is already present. If the entry is not in the table, it is added and the reference indicator is created as a blank. If the entry is in the table, the reference indicator is changed to M to indicate a multi-defined file and a message is printed out. Each entry consists of four words in the following format:

Word	1	2	3		4	
Bits	0 - 15	0 - 15	0 - 7	8 - 15	0 - 7	8 - 15
	Filename (in namecode)		Ref Indic*	File Type**	Sequence Number	

*Reference Indicator
- Blank - Unreferenced
- E - Referenced in Extension Specifications
- R - Referenced in Input Specifications
- O - Referenced in Output Specifications
- M - Multi-Defined Filename

**File Type

Bit	Value	Meaning
0	0 1	Not Index Sequential File Index Sequential File
1	0 1	Input or Update File Output File
2	0 1	ADD not needed on output for file ADD must be specified for file on output
3	0 1	Not Disk Disk
4	0 1	Extension Code Extension Code
5	0 1	Not a Chained File Chained File
6	0 1	Not an RA File RA File
7	0 1	Not a Table File Table File

TENT Table (TENT)

The TENT table (built by RG16, and used by RG16 and RG17) can contain as many as ten entries. Each entry is four words long, in the following format:

Word	Bits	Contents
1	0-15	Internal Sequence Number
2	0-7	File Type (I-O-U-C, in EBCDIC) RG16 changes this entry to: a) 08, if 1403 printer is used, b) OA, if 1132 printer is used, c) OC, if console printer is used.
	8-15	File Designation (P-S-C-R-T, in EBCDIC)
3	0-15	Record Length (in binary)
4	0-15	Key Length (in binary)

Input/Output Table (IOTAB)

This table is built from file description entries, and is used in generating I/O routines. This table can contain as many as ten entries. The code entries are right-justified, in hexadecimal notation. Each entry is 7 words long, in the following format:

WORD	CONTENTS	
	CODE	FILE TYPE
1	0010	1403 Printer
	12	1132 Printer
	14	Console Printer
	20	1442 Punch Output
	22	1442 Reader/Punch Output
	24	1442 Reader/Punch Combined
	26	1442 Reader/Punch Input
	28	2501 Reader Input
	40	Sequential Update
	42	Random Update
	46	Sequential Output
	48	ISAM LOAD
	4A	ISAM ADD
	4C	ISAM Sequential Update
	4E	ISAM Random Update
2	Record Length (in binary)	
3	Key Length (in binary)	
4-5	Symbolic Filename (in namecode)	
6	Overflow indicator, if printer; or number of sectors necessary, if ISAM LOAD file.	
7 (0-7) 7 (8-15)	Mode of processing column 28 for FDS (R,L) a) File length, if RA File b) Number of index entries per sector, if ISAM LOAD file (in binary)	

Note: A 01 is ORed into the Code entry if dual I/O is requested.

Control Level Address Table

The Control Level Address Table (built by RG17, for RG38) consists of nine one-word entries. Each entry contains the address of one of the nine control level fields. The format of these entries is:

Word	Contents
1	Address of 1st control level field
2	Address of 2nd control level field
3	Address of 3rd control level field
4	Address of 4th control level field
5	Address of 5th control level field
6	Address of 6th control level field
7	Address of 7th control level field
8	Address of 8th control level field
9	Address of 9th control level field

Overflow Table (SEQOF)

The Overflow Table (built by RG22, for RG54) contains two entries. Each entry is two words long, in the following format:

Word	Contents
1	Sequence number of the File Description Specification
2	Address of the Overflow indicator

Filel Table (FILE1)

The Filel Table (built by RG22) may have from one to ten entries, depending upon the number of files. Each entry is three words long, in the following format:

Word	Bit	Contents
1	0-15	Address of I/O Routine
2	0-15	Address of FILTAB-4 for this file
3	0-3	Must be zero for Primary and Secondary files; otherwise, bits 0-7 are C, R, T, or O.
	4	1 Primary File 0 Secondary File
	5	1 Ascending Matching Fields 0 Descending Matching Fields
	6	1 Sequence Check Required 0 Sequence Check Not Required
	7	1 EOF for this File does not count in EOJ 0 EOF for this File Counts in EOJ
	8-14	(Not used)
	15	1 Open/Close routine required 0 Open/Close routine not required

Note: Word 1 is filled in by RG24, RG26, or RG28; word 2 is filled in by RG42; word 3 is filled in by RG22.

This area is in the Resident phase, between the labels 'ZRDSP' and 'COEND'. It provides addresses and constants used by the compiler. Each field is one word long, except 'NOTES' which is 17 words long. The fields and their contents are:

FIELD	DISPLACEMENT Dec.	Hex.	CONTENTS	USED BY Phases
ZRDSP	+0	00	• Address of read source statements	00, 02, 04, 06, 08
			• After Enter phases, it is used by DSF routine for the Patch Address	32, 36, 38, 40, 42, 46, 52, 58
			If it contains a 0, there is no patch; if a non-zero, it contains the patch address.	
ZPRSP	+1	01	• Address of List Source Statements Routine	00, 02, 04, 06, 08
			• Address of Print Listing Routine (PRTSP)	04, 10, 12, 14, 16, 17, 19, 58, 60
ZGTCM	+2	02	• Address of Get Compression Routine (GETCM)	10, 12, 14, 16, 17, 36, 38, 40, 42, 44, 46, 52, 54
ZPTCM	+3	03	• Address of Put Compression Routine (PUTCM)	02, 04, 06, 08
ZPTOB	+4	04	• Address of Put Object Code Routine	10, 12, 24, 26, 28, 38, 42, 52, 54, 58
			• Put out DSF Code	14, 32, 34, 38, 40, 42, 44, 46
ZPBUF	+5	05	• Address of the Principal Print Buffer	02, 04, 06, 08, 10, 12, 14, 16, 17, 19, 58, 60
			• Address of the FILE1 Table	22, 24, 26, 28, 32, 34, 42, 58, 54
			• Address of Table Area	
ZRBUF	+6	06	• Address of Principal Read Buffer	00, 02, 04, 06, 08
			• After Enter Phases it contains the number of sectors per write of DSF code in working storage.	
ZPTER	+7	07	• Address of Error Note Routine (PRTER)	00, 02, 04, 06, 08, 10, 17, 14, 16, 19, 12

Table 3. Communications Area (COMAREA) (Part 1 of 6)

FIELD	DISPLACEMENT Dec.	Hex.	CONTENTS	USED BY Phases
ZOBUF	+8	08	• Address of Object Code Buffer	PUTOB
ZOEND	+9	09	• Address of End of Object Code Buffer +1	PUTOB
ZCBUF	+10	0A	• Address File Description Compression .... • Address of Compression Buffer 1 .... • Address of First Block of Compression	22 32 34, 38
ZCBF2	+11	0B	• Address of Compression Buffer 2 .... • Address of Compression Block	36, 42, 58 38
ZNTCM	+12	0C	• Address of Next Buffer Word .... • Address of Compression Area .... • Address of Next Compression Word .... • Address of Current Compression Specification	04, 10, 12, 44 02, 06, 08 46 52, 54
ZACNT	+13	0D	• Address Counter .... • Object Code Location Counter .... • Last Object Code Address	10, 12, 22, 26, 28, 32, 34, 38, 40, 42, 52, 54, 58 17, 24, 36, 44, 46, 14 60
ZTBAD	+14	0E	• Address of RPG Option Word (Column 11 of RPG Control Card).	
ZSEQI	+15	0F	• Sequence number of first Input Specification	04, 10
ZSEQC	+16	10	• Sequence number of first Calculation Specification	06, 17
ZSEQO	+17	11	• Sequence number of first Output Specification	08, 17
ZSEQL	+18	12	• Sequence number of last Output Specification .... • Statement Sequence number	04 02, 06, 08
ZPHCL	+19	13	• Address of Get Next Phase Routine (CALPH)	02, 04, 06, 10, 12, 16, 17, 19, 22, 24, 26, 28, 32, 34, 36, 38, 40, 42, 44, 46, 52, 54, 58
ZCALS	+20	14	• Literal Usage Switch .... • Indicator Word .... • Cal switch .... • Contains Switches set by RG06 and RG08.	14 06, 08 04, 12
ZBLIN	+21	15	• Compression block number of first Input Specification ..	04, 10, 34, 36, 38, 40, 42
ZBLCA	+22	16	• Compression block number of first Calculation Specification	06, 14, 44, 46

Table 3. Communications Area (COMAREA (Part 2 of 6)

FIELD	DISPLACEMENT Dec.	Hex.	CONTENTS	USED BY Phases
ZBLOT	+23	17	• Compression block number of first Output Specification .	08, 14, 52, 54
(The term "block" usually refers to a buffer which has been written on a disk. Block 1, which always remains in core, is the exception.)				
ZIPTR	+24	18	• Address of first Input Resulting Indicator	10
ZLST	+25	19	• Address of last indicator.	
ZNOIN	+26	1A	• Number of Input indicators	10, 58
ZINAD	+27	IB	• Address of First Input Specification in a Compression block	04, I0, 36, 38, 40, 42
ZCALA	+28	1C	• Address of first Calculation Specification	06, 14, 17, 44, 46
ZOUTA	+29	1D	• Address of first Output Specification in a Compression block	08, 14, 17, 52, 54,
ZBLOC	+30	1E	• Number of current compression block.	04, 06, 08, 10, 12, 14, 17, 36, 38, 42, 44, 46, 52, 54
ZLSTB	+31	1F	• Number of Last compression block.	
ZM1LN	+32	20	• Length of M1	16, 40, 42
ZM2LN	+33	21	• Length of M2	16
ZM3LN	+34	22	• Length of M3	16
ZM4LN	+35	23	• Length of M4	16
ZM5LN	+36	24	• Length of M5	16
ZM6LN	+37	25	• Length of M6	16
ZM7LN	+38	26	• Length of M7	16
ZM8LN	+39	27	• Length of M8	16, 54, 58
ZM9LN	+40	28	• Length of M9	16, 54, 58
ZC1LN	+41	29	• Length of C1	16, 17

Table 3. Communications Area (COMAREA) (Part 3 of 6)

FIELD	DISPLACEMENT Dec.	Hex.	CONTENTS	USED BY Phases
ZC2LN	+42	2A	• Length of C2	16, 17
ZC3LN	+43	2B	• Length of C3	16, 17
ZL1LN	+44	2C	• Length of L1 (The addresses of each of the control level indicator fields are found at address 38D HEX.)	16, 17
ZL2LN	+45	2D	• Length of L2	16, 17
ZL3LN	+46	2E	• Length of L3	16, 17
ZL4LN	+47	2F	• Length of L4	16, 17
ZL5LN	+48	30	• Length of L5	16, 17
ZL6LN	+49	31	• Length of L6	16, 17
ZL7LN	+50	32	• Length of L7	16, 17
ZL8LN	+51	33	• Length of L8	16, 17
ZL9LN	+52	34	• Length of L9	16, 17, 38
ZSTR1	+53	35	• Sterling Input Option	04, 36
ASTR2	+54	36	• Sterling Output Option	52
ZINVR	+55	37	• RPG Inverted Print Option	06, 14
BEGWS	+56	38	• Address of disk working storage	60
ZFLNM	+57	39	• Address of Filename Table	02, 06, 10, 22
ZALTS	+58	3A	• RPG Alternate Collating Sequence Option	40
BEGOB	+59	3B	• Beginning of DSF program • Address of Working storage of first block of DSF code written on disk	00, 60 00, 60
ZNEWH	+60	3C	• Location counter switch • New label switch for DSF routine • Address of New Heading needed	24 10, 40, 52 32
ZERCD	+61	3D	• Error codes for Wrap-up phase 0 - Normal Compilation 1 - Working Storage Exceeded 2 - Serious Compilation Error 4 - Diagnostic run error • Completion Code	00, 19 60

NOTES: 62-78 (DEC) 3E-4E (HEX) The contents of this area change after RG22 is given control.

Table 3. Communications Area (COMAREA) (Part 4 of 6)

Each time a phase prior to the Error Message Phase (RG 19, 20, 21) detects an error, it orders the Resident phase to print the note number identifying the error. At this time a bit is set in the error note area, NOTES. This area is a block of 17 words (272 bits). Each bit in the block is a switch associated with one of the 272 possible error notes. Bit 0 of word 0 represents note 15, Bit 15 of word 0 represents note 0, etc., as follows:

	Bits Position															
Displacement from NOTES	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
+2	47															32
+3	63															
+4	79															
+5	95															
+6	111															
+7	127															
+8	143							ERROR NOTE IDENTIFIERS INDICATORS								
+9	159															
+10	175															
+11	191															
+12	207															
+13	223															
+14	239															
+15	255															

Table 3. Communications Area (COMAREA) (Part 5 of 6)

If no bits are set, phase RG22 is loaded. If bits are set in the NOTE area, each bit set will be tested and appropriate error messages printed.

		Phases RG24-RG58 use the 17 word NOTES area to save addresses, as follows: NOTES +0 - address of Table Dump routine NOTES +1 - address of Detail lines NOTES +2 - address of Detail Lines Table NOTES +3 - address of Total Lines NOTES +4 - address of Total Lines Table NOTES +5 - (Not Used) NOTES +6 - address of return address for Except lines NOTES +7 - address of Detail Calcs NOTES +8 - address of Total Calcs NOTES +9 - address of Chain Routine NOTES+10 - (Not Used) NOTES+11 - (Not Used) NOTES+12 - address of Record Address routine NOTES+13 - address of Table Load routine NOTES+14 - If a Record Address File, the address of the "To File" name NOTES+15 - address of Control Field routine NOTES+16 - address of the Low Field Control Block (LFDAD--used by RG40)
ZBLCT	+79 x '4F'	• Address of the Wrap-up routine in RG10, which is linked to from RG60; when control returns to RG60, it contains the Disk Block Count.
COEND		• This label identified the end of the COMAREA.

Table 3. Communications Area (COMAREA) (Part 6 of 6)

This section contains two tables designed as serviceability aids for maintaining or modifying the 1130 RPG Compiler program: an External Reference Table and a Table of Control Block and Table Usage.

EXTERNAL REFERENCE TABLE

This table summarizes external references for the phases of the 1130 RPG compiler. In this publication, an external reference is a linkage from the calling (or linking) phase to another common routine, which returns control to the next instruction following the linkage. (Note that the called phase may have external references of its own.) A situation in which the called routine does not return control to the calling phase is not considered an external reference.

The vertical entries are divided into three categories:

1. Routines Referenced.
2. Constants and addresses defined in the Monitor Communication area (COMMA).
3. Constants and addresses defined in the Resident Phase Communication Area (COMAREA).

The vertical entries are listed alphabetically; the horizontal entries (module and routine names) are listed numerically.

	MODULE	RG00							RG02	RG04	RG06	RG08	RG10			RG12	RG14	RG16	RG17	RG19	RG22	RG24	RG26	RG28	RG32	RG34	RG36	RG38	RG40	RG42	RG44	RG46	RG52	RG54	RG58	RG60
	ROUTINE		PRTER	CALPH	PUTCM	RDSPC	GETCM	PRTSP						OBEND	PUTOB																					
ROUTINES REFERENCED	CALPH	X				X			X	X	X		X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
	DZ000	X		X	X		X								X																					X
	GETCM						X			X			X			X	X	X	X								X	X	X	X	X	X	X	X		
	OBEND																																			X
	PRTER	X							X	X	X	X	X					X	X																	
	PRTLN	X				X		X							X																				X	X
	PRTSP	X	X						X	X	X	X	X			X	X	X	X	X																X
	PUTCM					X			X	X	X	X																								
	PUTOB												X			X	X		X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	
	RDSPC	X							X	X	X	X																								
	$DBSY	X													X																					X
	$EXIT	X				X																														X
COMMA	$CORE	X																																		
	$CTSW	X																																		
	$FPAD	X																																		X
	$IBSY	X																																		
	$IBT4	X																																		
	$KCSW	X																																		
	$LAST	X																																		
	$NDUP	X																																		X
	$NEXQ	X																																		X
	$PBSY	X																																		
	$PHSE*	*																																		*
	$WSDR	X																																		X

* $PHSE is used by every phase

Table 4. External Reference Table (Part 1 of 2)

MODULE		RG00							RG02	RG04	RG06	RG08	RG10			RG12	RG14	RG16	RG17	RG19	RG22	RG24	RG26	RG28	RG32	RG34	RG36	RG38	RG40	RG42	RG44	RG46	RG52	RG54	RG58	RG60
ROUTINE			PRTER	CALPH	PUTCM	RDSPC	GETCM	PRTSP						OBEND	PUTOB																					
COMAREA	BEGOB																																			X
	BEGWS																																			X
	LFDAD																												X							
	NOTES																			X						X		X							X	
	ZACNT												X		X	X	X					X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	ZALTS																												X							
	ZBLCA																X														X	X				
	ZBLCT																																			X
	ZBLIN									X			X														X	X	X	X						
	ZBLOC									X		X	X			X	X										X	X		X	X	X	X	X		
	ZBLOT											X					X																X	X		
	ZCALA										X						X														X	X				
	ZCALS									X	X	X				X	X																			
	ZCBF2																										X	X		X					X	
	ZCBUF																				X				X	X										
	ZERCD																			X																X
	ZFLNM								X		X										X															
	ZINAD									X			X														X	X	X	X						
	ZINVR										X						X																			
	ZIPTR												X																						X	
	ZLST												X																							
	ZL9NL																											X								
	ZM1LN																												X	X						
	ZM8LN																																	X		
	ZM9LN																												X					X		
	ZNEWH														X							X			X				X							
	ZNTCM								X	X	X	X	X			X															X	X	X	X		
	ZNOIN												X																						X	
	ZOUTA											X					X																X	X		
	ZPBUF								X	X	X	X	X			X	X			X	X	X	X	X	X					X				X	X	X
	ZRBUF								X	X		X																								
	ZRDSP								X						X												X	X	X	X		X				
	ZSEQC									X	X																									
	ZSEQI									X			X																							
	ZSEQL								X	X	X	X																								
	ZSEQO									X	X	X																								
	ZSTR1									X																	X									
	ZSTR2																																X			
	ZTBAD											X																								

Table 4. External Reference Table (Part 2 of 2)

CONTROL BLOCK AND TABLE USAGE

This table contains a list of control blocks and tables created by the 1130 RPG compiler. It also names the routines that create and modify them.

CONTROL BLOCK OR TABLE	BUILT BY:	MODIFIED BY:
Filename Table (FLENM)	RG02	RG02 RG04 RG06 RG08 RG22 RG58
Tent Table (TENT)	RG16	RG16 RG17
Input/Output Table (IOTAB)	RG22	RG22 RG24 RG26 RG34
Control Level Address Table	RG17	RG38
Overflow Table (SOSOF)	RG22	RG54
FILE1 Table (ZPBUF)	RG22 RG24 RG26 RG28	RG58 RG42 RG40
Communications Area (COMAREA)	This area is explained in detail in Table 3, because it is accessed by every phase.	

Table 5. Control Blocks and Tables Created by the 1130 RPG Compiler

This section is composed of tables illustrating how information from the various specification sheets used by RPG is put into compressed format. By eliminating unnecessary information such as blanks, RPG can reduce I/O operations, save storage space, and still retain important information in main storage via compression. The presence or absence of certain compression information is indicated in the α-word. For example, in a calculation compression, if there is no control level specified, bit Ø of the α-word will be off and the position of any information following it will be decremented by its length (in this case 1 word).

The following compression formats are described in this section:

- File Description
- Extension
  - Record Address Files
  - Chain Files
  - Table Entries
- Input
  - Record Type
  - Field Type
- Calculation
- Output-Format
  - Record Type
  - Field Type

In each of the compression format tables, a character string is used to illustrate the grouping of information in that format. The meaning of the field of information represented by each character in the string is then explained. The length and location of each field within the compression is shown by entries in the "Word" and "Bits" columns to the left of the character. References to columns identify which column of the corresponding specification sheet would contain the specified entry. At the end of each compression description, the minimum and maximum lengths of the compression are noted by the "Minimum" and "Maximum" entries.

FCαNTDBLCRPOI		
Word	**Bits**	
1	0-7 8-15	F Type of Specification (EBCDIC F) C Length of compressed specification (in binary)
2	0-15	α α- word **Bit** — **Value** — **Description** 1 — 0 — No E in column 17 1 — E in column 17 2 — 0 — Ascending sequence (column 18) ascending sequence is 1 — Descending sequence assumed if no entry is made 3 — 0 — No extension code 1 — Extension code 4 — 0 — ISAM add not specified 1 — ISAM add specified 5 — 0 — Not ISAM load 1 — ISAM load 6 — 0 — 0 or 1 entered for bit 2, above 1 — No entry for bit 2
3	0-7 8-15	(Not used) N Sequence Number of this File Description Specification
4	0-7 8-15	T File Type (column 15 in EBCDIC) D File Designation (column 16 in EBCDIC)
5	0-15	B (Not used)
6	0-15	L Record Length (columns 24-27)
7	0-15	C Device Hex Code — Device 0002 — READ 42 04 — READ 01 06 — PUNCH 42 08 — PRINT 03 0A — PRINTER 0C — CONSOLE 0E — DISK
8	0-15	R a) Length of RA File Field (columns 29-30) b) Length of Key Field (columns 29-30) c) Overflow Indicator (columns 33-34 in EBCDIC)
9	0-7 8-15	P Mode of Processing (column 28 in EBCDIC) 0 Type of File Organization (column 32 in EBCDIC)
10-12	0-15	I For ISAM load, maximum file size, left-justified in EBCDIC (columns 47-51)

Note: Entries in words 8-12 are optional. If an optional word is required, all prior optional words (required or not) must be included in the compression specification and unused words must be padded with a fill character. Minimum - 7 words; maximum - 12 words.

Table 6. File Description Compression

RA File Entries

RCFT

Word	Bits	
1	0-7 8-15	R Identification Code (EBCDIC R) C Length of Compressed Specification (in binary)
2	0-7 8-15	F From Filename (Sequence Number of File Description Entry) T To Filename (Sequence Number of File Description Entry)

Note: Word 2 comes from the Filename Table. Minimum - 2 words; Maximum - 2 words.

Chain File

ECNSF

Word	Bits	
1	0-7 8-15	E Identification Code (EBCDIC E) C Length of Compressed Specification (in binary)
2	0-15	N Chaining field number (columns 9-10)
3	0-15	S Record Sequence of the chaining file (columns 7-8)
4	0-7 8-15	F From Filename (Sequence number of File Description Entry) T To Filename

Note: Word 3 comes from the Filename Table. Minimum - 4 words; Maximum - 4 words.

Table Entries

TCFOQNBELPDA∝UVWXYZ

Word	Bits	
1	0-7 8-15	T Identification Code (EBCDIC T) C Length of Compressed Specification (in binary)
2	0-7 8-15	F From Filename (Sequence Number of File Description Entry) O To Filename (Sequence Number of File Description Entry), blank if not specified
3-5	0-15	Q Blanks, to later contain generated machine address
6-7	0-15	N Table name (columns 30-32)
8	0-15	B Number of entries per record (columns 33-35)
9	0-15	E Number of entries per table (columns 36-39)
10	0-15	L Length per entry (columns 40-42)
11	0-7 8-15	P Pack indicator (column 43) D Numeric indicator (column 44)
12	0-7 8-15	A Sequence indicator (column 45) ∝ Second table indicator; 00 if no second table, F0 if second table
13-15	0-15	U Blanks; same entry as words 3-5
16-17	0-15	V Table name same entry as words 6 and 7 (columns 49-51)
18	0-15	W Number of entries per record; same entry as word 10 (columns 52-54)
19	0-7 8-15	X Pack Indicator; same entry as word 11, bits 0-7 (column 55) Y Numeric Indicator; same as word 11, bits 8-15 (column 56)
20	0-7 8-15	Z Sequence Indicator; same as word 12, bits 0-7 (column 57) (Not used)

Note: Words 13-20 are optional. Word 2 (bits 0-15) comes from Filename Table. Minimum - 12 words, when ∝ is set at 00; Maximum - 20 words, when ∝ is set at F0.

Table 7. Extension Compression

Record Type		
IΔ∝FQRSPTCAVWXYZ		
Word	**Bits**	
1	0-7 8-15	I Identification Code (EBCDIC I) Δ Length of Compressed Specification (in binary)

2	0-15	∝ ∝- word
		Bit / Value / Meaning
		0 / 0 / No stacker select
		  / 1 / Stacker select
		1-2 / 00 / OR type record
		  / 01 / AND type record
		  / 10 / Alphabetic Sequence
		  / 11 / Numeric Sequence Check
		3 / 0 / Numeric Mandatory Record
		  / 1 / Numeric Optional Record
		4 / 0 / Numeric 1 or more records
		  / 1 / Numeric 1 only record
		5 / 0 / Filename not specified
		  / 1 / Filename specified
		6-7 / 00 / No record codes
		  / 01 / 1 record code
		  / 10 / 2 record codes
		  / 11 / 3 record codes
		8-15 / / (Not used)

3	0-15	F Sequence number of File Description Specification, from Filename Table
4	0-15	Q Sequence of Input Record Type (columns 15-16)
5	0-15	R Resulting Indicator code (columns 19-20)
6	0-15	S Stacker Select (column 42)
7	0-15	P Location of character in record format (columns 21-24)
8	0-7	T Type of test Bit / Value / Meaning 0 / 1 / Negative character test, otherwise positive 1 / 1 / Character test, otherwise not 2 / 1 / Zone test, otherwise not 3 / 1 / Digit test, otherwise not 4-7 / 0 /
	8-15	C Character in test (column 27)
9	0-15	A Location of character in record format (columns 28-31)
10	0-7 8-15	V Type of test; same as word 8 (columns 32-33) W Character in test (column 34)

Table 8. Input Compression (Part 1 of 3)

Word	Bits		
11	0-15	X	Location of character in record format (columns 35-38)
12	0-7 8-15	Y Z	Type of test; same as word 8 (columns 39-40) Character in test (column 41)

Note: Words 6-12 are optional. Minimum - 5 words; maximum - 12 words.

Table 8. Input Compression (Part 2 of 3)

<table>
<tr><td colspan="4">Field Type

DΔ∝FXPANLM/CRPIZS</td></tr>
<tr><th>Word</th><th>Bits</th><th></th><th></th></tr>
<tr><td>1</td><td>0-7
8-15</td><td>D
Δ</td><td>Identification Code (EBCDIC D)
Length of Compressed Specification (in binary)</td></tr>
<tr><td>2</td><td>0-15</td><td>∝</td><td>∝- word
<table>
<tr><th>Bit</th><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>No CTL level specified</td></tr>
<tr><td></td><td>1</td><td>CTL level specified</td></tr>
<tr><td>1</td><td>0</td><td>No matching field specified</td></tr>
<tr><td></td><td>1</td><td>Matching field specified</td></tr>
<tr><td>2</td><td>0</td><td>No field record relation specified</td></tr>
<tr><td></td><td>1</td><td>Field record relation specified</td></tr>
<tr><td>3</td><td>0</td><td>Plus not used</td></tr>
<tr><td></td><td>1</td><td>Plus used</td></tr>
<tr><td>4</td><td>0</td><td>Minus not used</td></tr>
<tr><td></td><td>1</td><td>Minus used</td></tr>
<tr><td>5</td><td>0</td><td>Zero not used</td></tr>
<tr><td></td><td>1</td><td>Zero used</td></tr>
<tr><td>6</td><td>0</td><td>No sterling</td></tr>
<tr><td></td><td>1</td><td>Sterling</td></tr>
<tr><td>7</td><td>0</td><td>No chaining</td></tr>
<tr><td></td><td>1</td><td>Chaining</td></tr>
<tr><td>8-15</td><td></td><td>(Not used)</td></tr>
</table></td></tr>
<tr><td>3</td><td>0-15</td><td>F</td><td>From position in binary (columns 44-47)</td></tr>
<tr><td>4</td><td>0-15</td><td>X</td><td>Length of field in binary (columns 44-51)</td></tr>
<tr><td>5</td><td>0-7
8-15</td><td>P
A</td><td>Packed Indicator (column 43)
Decimal position/blank for Alpha fields (column 52)</td></tr>
<tr><td>6-8</td><td>0-15</td><td>N</td><td>Field name (columns 53-58)</td></tr>
<tr><td>9</td><td>0-7
8-15</td><td>L
M
C</td><td>Control level (column 60 in EBCDIC)
Matching value (column 62, or
Chaining value (column 62 in EBCDIC)</td></tr>
<tr><td>10</td><td>0-15</td><td>R</td><td>Field record relation indicator (columns 63-64)</td></tr>
<tr><td>11</td><td>0-15</td><td>P</td><td>Plus field indicator (columns 65-66)</td></tr>
<tr><td>12</td><td>0-15</td><td>I</td><td>Minus field indicator (columns 67-68)</td></tr>
<tr><td>13</td><td>0-15</td><td>Z</td><td>Zero field indicator (columns 69-70)</td></tr>
<tr><td>14-15</td><td>0-15</td><td>S</td><td>Sterling field (columns 71-74)</td></tr>
</table>

Note: Words 9-15 are optional. Minimum - 8 words; maximum - 15 words.

Table 8. Input Compression (Part 3 of 3)

CΔ∝BII$_1$I$_2$FOTRLDHSMZ		
Word	Bits	
1	0-7 8-15	C Identification Code (EBCDIC C) Δ Length of this compressed specification

2	0-15	∝ ∝- word
		
		Bit / Value / Meaning
		0 / 0 / No control level
		  / 1 / Control level
		1 / 0 / No indicators
		  / 1 / Indicators
		2 / 0 / No Factor 1
		  / 1 / Factor 1
		3 / 0 / Factor 1 is a Field Name
		  / 1 / Factor 1 is a literal (reserve space in Literal Compression format)
		4 / 0 / Factor 2 is a Field Name
		  / 1 / Factor 2 is a literal (reserve space in Literal Compression format)
		5 / 0 / No Plus indicator
		  / 1 / Plus indicator
		6 / 0 / No Minus indicator
		  / 1 / Minus indicator
		7 / 0 / No Zero indicator
		  / 1 / Zero indicator
		8-15 / / (Not used)

3	0-15	B Control level
4-5	0-15	I Indicator (columns 7-8)
6-7	0-15	I$_1$ Indicator (columns 9-11)
8-9	0-15	I$_2$ Indicator (columns 12-14)
10-15	0-15	F Factor 1 (columns 18-24) (If Field Name, 3 words long; if literal, 6 words)
16	0-15	O Operation Code* (columns 28-32)
17-22	0-15	T Factor 2 (columns 33-42) (If Field Name, 3 words long; if literal, 6 words)
23-25	0-15	R Result Field
26	0-15	L Length of field in binary format
27	0-7 8-15	D Decimal positions H Half-adjust--blank if none
28	0-15	S Plus-High indicator
29	0-15	M Minus-Low indicator
30	0-15	Z Zero-Equal indicator

Note: Minimum - 11 words; maximum - 30 words.

Table 9. Calculation Compression (Part 1 of 2)

*Operation Codes

CODE	ENTRY (IN HEXADECIMAL)	
	Bits 0-7	Bits 8-15
ADD	F1	FA
BEGSR	F0	02
CHAIN	F0	00
COMP	F4	04
DIV	F4	FD
ENDSR	F0	03
EXCPT	F0	01
EXIT	F3	FE
EXSR	F0	04
GOTO	F3	01
LOKUP	F5	01
MHHZO	F6	06
MHLZO	F6	05
MLHZO	F6	04
MLLZO	F6	03
MOVE	F6	01
MOVEL	F6	02
MULT	F4	FC
MVR	F4	03
RLABL	F9	FF
SETOF	F2	00
SETON	F2	F0
SUB	F1	FB
TAG	F7	01
TESTZ	F5	02
Z-ADD	F1	03
Z-SUB	F1	04

Table 9. Calculation Compression (Part 2 of 2)

Record Type		
		O∆∝NFSBADKII$_1$I$_2$
Word	**Bits**	
1	0-7	O Identification Code (EBCDIC O)
	8-15	∆ Length of this compressed specification
2	0-15	∝ ∝ - word
		Bit / Value / Meaning: 1 / 0 / No heading or detail line 1 / 1 / Heading or detail line 2 / 0 / No except lines 2 / 1 / Except lines 3 / 0 / No total lines 3 / 1 / Total lines 4 / 0 / No conditioned overflow 4 / 1 / Conditioned overflow 5-6 / 00 / Filename present 5-6 / 01 / OR type* 5-6 / 10 / AND type** 7-8 / 00 / No resulting indicators 7-8 / 01 / 1 resulting indicator 7-8 / 10 / 2 resulting indicators 7-8 / 11 / 3 resulting indicators 9 / 0 / No space 9 / 1 / Space 10 / 0 / No skip before 10 / 1 / Skip before 11 / 0 / No skip after 11 / 1 / Skip after 12-15 / (Not used)
3	0-15	N Internal Sequence Number (in binary)
4	0-7	F Sequence number from File Description Specifications
	8-15	S Stacker Select (blank, 1, or 2)
5	0-7	B Space Before
	8-15	A Space After
6	0-15	D Skip Before
7	0-15	K Skip After
8-9	0-15	I Resulting Indicator
10-11	0-15	I_1 Resulting Indicator
12-13	0-15	I_2 Resulting Indicator

Note: Words 5-13 are optional. Minimum - 4 words; maximum - 13 words.

Table 10. Output-Format Compression (Part 1 of 4)

*OR Type $O\Delta\propto NSBADKII_1I_2$ Note that F is omitted Minimum - 3 words Maximum - 12 words
**AND Type $O\Delta\propto NII_1I_2$ Minimum - 3 words Maximum - 9 words

Table 10. Output-Format Compression (Part 2 of 4)

Field Type		
		$M\Delta\alpha TEFII_1I_2SNCLXYZBP$
Word	**Bits**	
1	0-7 8-15	M Identification Code (EBCDIC M) Δ Length of this compressed specification (in binary)
2	0-15	α α- word Bit / Value / Meaning 0-1 / 00 / No output indicator 01 / 1 output indicator 10 / 2 output indicators 11 / 3 output indicators 2 / 0 / No field name 1 / Field name 3 / 0 / Constant 1 / Edit word 4 / 0 / No special edit code 1 / Special edit code 5 / 0 / No blank after printing 1 / Blank after printing 6 / 0 / No packed output 1 / Packed output 7 / 0 / No sterling 1 / Sterling 8 / 0 / Not PAGE field 1 / PAGE field 9-15 / (Not used)
3	0-15	T Internal sequence number
4	0-15	E Rightmost position of field (in binary)
5-7	0-15	F Field name
8-9	0-15	I Output Indicator
10-11	0-15	I_1 Output Indicator
12-13	0-15	I_2 Output Indicator
14-15		S Space allowed for literals and edit words; used by later phases
16	0-15	N Length of literal or edit word
17	0-15	C Fill character, if edit word
18-29	0-15	L Literal* or edit word
30	0-7 8-15	X Length of edit word (X word) (in binary) Y Description of edit word

Table 10. Output-Format Compression (Part 3 of 4)

Word	Bits	
		Bit — Value — Meaning 8 — 0 — No asterisk protection or zero suppression in edit word 8 — 1 — Asterisk protection or zero suppression in edit word 9-11 — (Not used) 12 — 0 — No fixed $ 12 — 1 — Fixed $ 13 — 0 — No floating $ 13 — 1 — Floating $ 14 — 0 — No minus sign 14 — 1 — Minus sign 15 — 0 — No CR symbol 15 — 1 — CR
31	0-7 8-15	Z If 0, no CR or Minus sign, otherwise displacement to CR/Minus sign B Number of blanks in edit word
32-33	0-15	P Sterling sign position

Note: Minimum - 4 words; maximum - 33 words.

*Literal Format

LDBA

Word	Bits	
18	0-15	L Length of literal in binary (if negative, bit 0 set to 1; if positive, bit 0 set to 0)
19	0-15	D Decimal length of literal; if alphameric, leave blank
20-21 22-29	0-15 0-15	B Blanks A Literal, if alphameric

Note: Minimum-Maximum - 12 words.

Table 10. Output-Format Compression (Part 4 of 4)

This section describes the main routines of the RPG object program (those functions that are typical of every RPG object program).

The description begins with a generalized flowchart and narrative section, which illustrates the cycle of operations within the object program.

Next, the tables and work areas that contain information directly related to the flow of the object program are examined. This is followed by a description of each of the object program routines. Actual code from the program listings is used in many places, to clearly describe the functions of particular routines.

A core storage allocation map is presented to show the locations of the object program routines during execution of the program. To aid in understanding these separate routines and their relationships to each other, a trace of an object program is presented next.

Certain functions of the RPG object program, e.g., processing with an RA file, or processing by C1, C2, or C3 type chaining, need more than a cursory explanation. These functions are described following the sample object program trace.

The next section, Library Subroutines, describes each of the subroutines that may be called by an RPG object program. In each case, the narrative is accompanied by a flowchart which illustrates the logic of the routine.

The last section contains a core storage dump of an RPG object program and instructions which enable the reader to find where the RPG indicators, fields, literals, key routines, etc., are located.

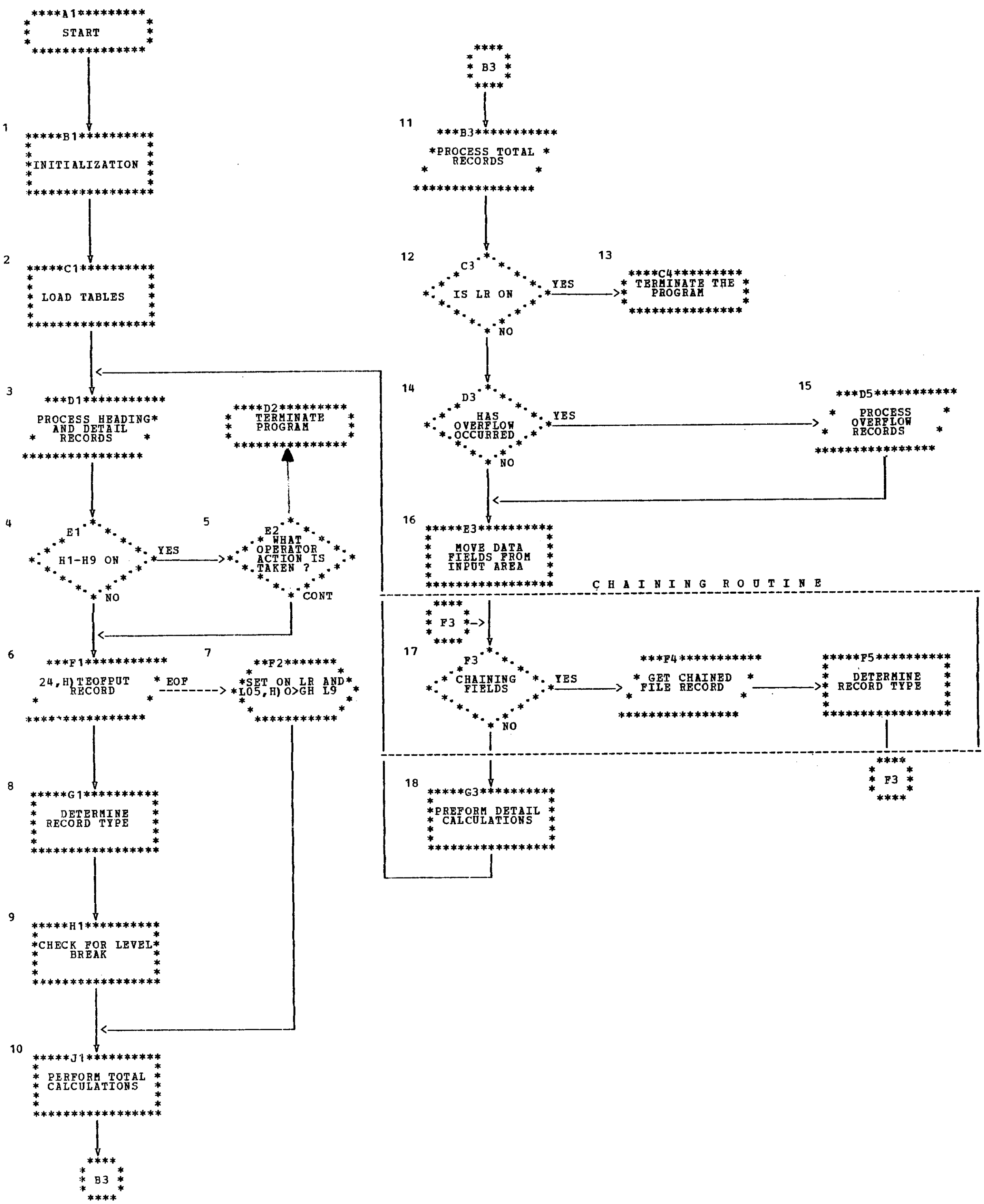


Chart GA. RPG Object Program (Simple Flow)

THE RPG OBJECT PROGRAM CYCLE

Each program generated by RPG uses the same general logic, and for each record to be processed, the program goes through the same general cycle of operations. To illustrate this concept, a generalized flowchart of an RPG object program is shown in Chart GA.

The following numbers correspond to the numbers on Figure 1. A program cycle begins with item 3 and continues through item 18.

1. Initialization (opening files, etc.) is performed.
2. Tables, if any are present, are loaded into core storage.
3. Before the first record is read, the program prepares and writes any heading information to be put out on the first page. After the first record has been read, the program prepares and writes heading and detail records which are not conditioned on overflow.

4&5. The halt indicators are tested. If any are on, the operator is notified and he may terminate or continue the job.

6. An input record is read into core storage.
7. If end-of-file occurs, the last record indicator (LR) is set on and all control-level indicators (L1-L9) are set on. The program branches to step 10.
8. Starting with line 1 of the Input Specifications sheets, and with the record just read, the program uses the record identification code to identify the record. When the identification code matches an entry on the Input Specifications sheet, the program turns on the resulting indicator that has been specified for the record.
9. If a control-field break has occurred, all appropriate control-level indicators are turned on.
10. Next, all total calculations are performed.
11. All total output records which are not conditioned on overflow are prepared and put out.

12.&13. The program tests for the last record indicator (LR). If it is on, the program is terminated.

14.&15. The program tests for an overflow condition. If overflow has occurred, total lines, heading lines, and detail lines (in that order) conditioned by overflow are printed.

16.&17. Data fields are extracted from the input record I/O area and moved into the assigned field areas. If any field is designated as a C1, C2, or C3 chaining field, the internal C1, C2, or C3 indicator is turned on and the chaining routine is given control. This routine retrieves the chained record.

18. Any detail calculations are performed and processing continues with item 3.

TABLES AND WORK AREAS

Before beginning a discussion of the object program routines, it may be helpful if certain object program tables and work areas are explained. The tables contain information which is directly related to the flow of the object program and will be examined in detail.

Function Address Table (FAT)

This 28 word table is generated for every RPG object program and contains the addresses of various RPG routines (Table 11). These addresses are compiled relative to location 0 and are relocated by the Core Load Builder.

File Input Tables (FITs)

These tables, one for each input file, are generated by RG42. Each table consists of four words of information for each record type within that file, plus a two word dummy entry at the end of the table. The four word entry contains identifying information associated with the record type that caused it to be generated and the two word dummy entry contains the address of the error routine (first word) for undetermined record types and asterisks (last word) to signify the end of the table.

Each record type would cause the following entry to be built (dummy entry is excluded):

Word:	1	2	3	4
Address of:	INPR	MFEXT	CLEV	INPF

Word1: This word contains the address of the INPR routine for the associated record type. The INPR routine performs a test on the characters of the input record to determine that the record is of the record type associated with this table entry. One INPR routine is generated for each input record type.

Word2: This word contains the address of the matching fields extraction routine (MFEXT) for this record type. MFEXT extracts the matching fields and holds them for comparisons and further processing.

Word	Label	Contents
1	ADSRT	Starting address of the RPG object program.
2	TABOT	Address of the Table Output routine.
3	HDAD	Address of Heading and Detail lines routine.
4	HDTAB	Address of Heading and Detail lines table (DTAB).
5	TAD	Address of Total lines routine.
6	TTAB	Address of Total lines table (TOTAB).
7	DCALC	Address of Detail Calculations.
8	TCALC	Address of Total Calculations.
9	CHAN1	Address of Chaining routine (for C1, C2, C3)
10		Not used.
11		Not used.
12		Not used.
13	RAFAD	Address of Record Address File routine.
14		Not used.
15		Not used.
16	TABLD	Address of Table Load routine.
17	CTLFD	Address of Control Field Compare routine.
18	LOWFD	Address of Low Field.
19	CHSAV	Save area for Chaining routine address.
20	EAD	Address of EXCPT lines routine.
21	RTE	Return address in calculations after EXCPT lines are executed.
22	ETAB	Address of EXCPT lines table (EXTAB).
23	ENDAD	Address of Close Files routine.
24		Not used.
25		Not used.
26		Not used.
27		Not used.
28		Not used.

Table 11. Contents of the Function Address Table

Word3: This word contains the address of the control-level extraction routine (CLEV) for this record type if this record type has control-level fields specified. This routine extracts control-level fields from the I/O area and moves them to the control-level hold area so that they may be tested for a control-level break.

If the associated record type has no control levels, word3 contains the address of TCLNK, an entry point to the Fixed Driver (see Fixed Driver for further information).

Word4: This word contains the address of the move input fields routine (INPF) for this record type.

To illustrate the File Input Table, assume the Input Specifications of a source program to be as follows:

IBM — International Business Machines Corporation — Form X21-9094, Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______ Page 01

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position (1)	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Field Name	Control Level (L1-L9)
01	I	CARDIN	AA			01	1		C	A				
02	I										1	10	FLD1	L1
03	I										11	20	FLD2	
04	I		BB			02	1		C	B				
05	I										8	15	FLD3	

The resulting File Input Table for record types 01 and 02 would contain the following:

	Word1	Word2	Word3	Word4
Entry1	Address of INPR routine for 01 record type	Address of MFEXT routine for 01 record type	Address of CLEV routine for 01 record type	Address of INPF routine for 01 record type
Entry2	Address of INPR routine for 02 record type	Address of MFEXT routine for 02 record type	Address of TCLNK in the Fixed Driver	Address of INPF routine for 02 record type
Dummy	Address of error routine for undetermined record type	/5C5C		

Entry1: These four words contain information associated with the record type identified by resulting indicator 01.

Word1 contains the address of the routine that tests the first position of an input record to see if it is an A (column 27).

Word2 points to the matching fields extraction routine for this record type. Since there are no fields designated by M1-M9 indicators, this routine will consist only of a branch instruction to continue processing.

Word3 points to the control-level extraction routine that will move FLD1 from the I/O area to the control-level hold area.

Word 4 contains the address of the routine that will move FLD1 and FLD2 from the I/O area to the field areas used for processing.

Entry2: These four words contain information similar to Entry1, but applicable to the record type associated with resulting indicator 02. Note that word3 for this record type contains the address of TCLNK because no control-level fields are specified.

Dummy: This two-word entry contains the address of the error routine (word1) for undetermined record types and always follows the last record type entry in the table. Word2 consists of asterisks signifying the end of this table.

Output Tables

The output tables used by the object program are generated by RG54. A maximum of four tables will be generated; one (DTAB) for heading and detail lines, one (TOTAB) for total lines, one (OTAB) for overflow lines, and one (EXTAB) for EXCPT lines.

DTAB

This table consists of one table entry for each heading and/or detail line specified on the Output-Format Specifications sheet. Each table entry consists of three words.

Word1: This word contains the address of the test output indicators routine. This routine tests the status of the indicators which condition the output line associated with this entry.

Word2: This word contains the address of the move output fields routine for this output line. This routine moves fields to be put out from the assigned fields area to the output buffer.

Word3: This word contains the address of the Input/Output Driver (IOD) routine associated with the file of which this line will be put out.

Note: Further information on IODs is contained under Object Time Routines.

The last entry in DTAB is a dummy three-word entry. The first word of this entry contains the address of the heading and detail lines wrap-up routine; a routine that sets up linkage to get the next record. Words 2 and 3 are not used.

TOTAB

This table consists of one table entry for each total line specified in the Output-Format Specifications sheet. Each table entry consists of three words containing the same information as the DTAB table entries. A dummy three-word entry is the last entry in TOTAB. Word1 of the dummy entry contains the address of the total lines wrap-up routine (a test for the occurrence of overflow).

OTAB

This table consists of one table entry for each total overflow line followed by one table entry for each detail overflow line. Each table entry consists of three words containing the same information as DTAB and TOTAB table entries. Again, the last table entry in OTAB is a three-word dummy entry. Word1 of this dummy entry contains the address of the overflow lines wrap-up routine (provides linkage to move fields from the I/O area to the fields area).

EXTAB

This table consists of one table entry for each EXCPT line specified on the Output-Format Specifications sheet. Each table entry consists of three words containing the same information as the DTAB, TOTAB, and OTAB table entries. Again, the last table entry in EXTAB is a three-word dummy entry. Word1 of this dummy entry contains the address of the EXCPT lines wrap-up routine (obtains return address and branches to calculations).

To illustrate these tables, the following diagrams contain one entry each.

DTAB

	Word1	Word2	Word 3
Heading or Detail Line entry	Address of the Test Output Indicators routine	Address of the Move Output Fields routine	Address of the Input/Output Driver (IOD)
Dummy entry	Address of B0020 in the Central Output Driver (COD)	Not used	Not used

TOTAB

	Word1	Word2	Word3
Total Line entry	Address of the Test Output Indicators routine	Address of the Move Output Fields routine	Address of the Input/Output Driver (IOD)
Dummy entry	Address of B0025 in the Central Output Driver (COD)	Not used	Not used

OTAB

	Word1	Word2	Word3
Total or Detail Overflow Line entry	Address of the Test Output Indicators routine	Address of the Move Output Fields routine	Address of the Input/ Output Driver (IOD)
Dummy entry	Address of B0030 in the Central Output Driver (COD)	Not used	Not used

EXTAB

	Word1	Word2	Word3
EXCPT Line entry	Address of the Test Output Indicators routine	Address of the Move Output Fields routine	Address of the Input/ Output Driver (IOD)
Dummy entry	Address of B0033 in the Central Output	Not used	Not used

Low Field, PS, and Processing Blocks

The Low Field four-word control block is generated in every object program and is modified throughout the execution of that object program. Each time a primary or secondary record is selected for processing, Low Field is filled with the following information about that record:

	Word1	Word2	Word3	Word4
Low Field	Address of the GET routine	Address of the CLEV routine	Address of the INPF routine	Address of the resulting indicator

Word1: When a primary or secondary record is selected for processing, the address of the GET routine for the file that contained that record is placed in the first word of Low Field. This address will be used on the next cycle of the object program at get next record time.

Word2: After a record has been selected for processing, the address of the control-level extraction routine for that record type is entered in Word2 of Low Field.

Word3: This word contains the address of the move input fields routine for the associated record type.

Word4: This word contains the address of the resulting indicator associated with this record type.

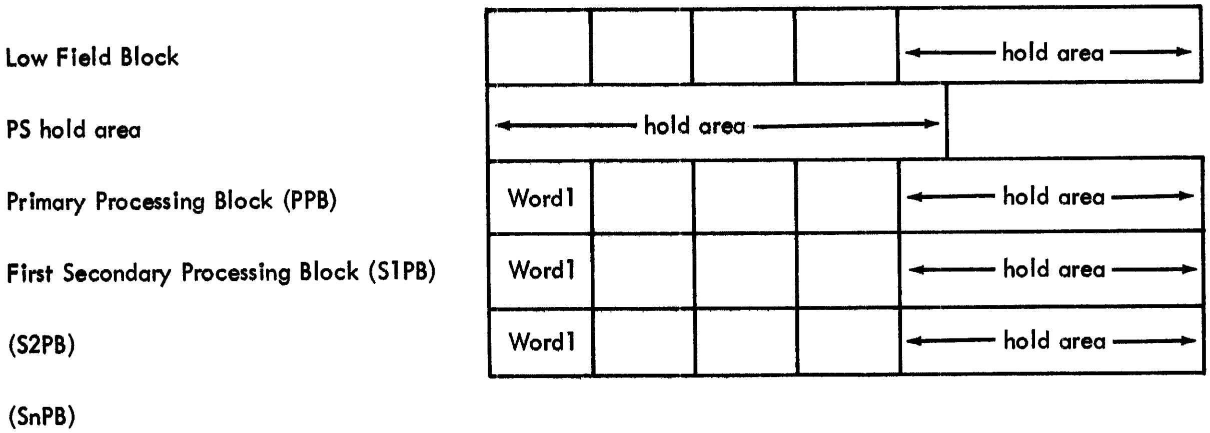
When matching fields are specified in the RPG source program, Low Field may be extended to include two hold areas to aid in processing the matching fields. Low Field plus the first hold area now becomes the Low Field Block, and the second hold area is named the PS hold area which contains the previous primary record.

K——Low Field Block——>|

Word1	Word2	Word3	Word4	Hold Area
PS Hold Area				

If no secondary files are present in the program and the primary file contains no M1-M9 fields, only Low Field will be generated. If no secondary files are present but the primary file does contain M1-M9 fields, the Low Field Block and the PS hold area are generated.

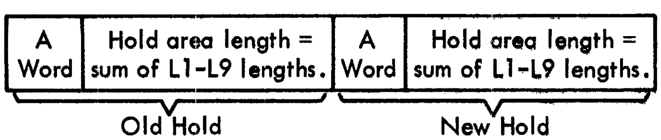
If both primary and secondary files are present, the Low Field Block and the PS hold area are joined by a Processing Block for each file. These Processing Blocks are generated as follows:



Word1 of each Processing Block contains the address of the GET routine for the associated file. Words 2-4 are dynamically filled with the information described for Words 2-4 of Low Field. Further information on the functions of Low Field, PS, and the Processing Blocks is contained in the Processing Multiple Input Files section of this publication.

Control Level Hold Areas

When fields are specified with control-level indicators in the RPG source program, two contiguous hold areas are generated. These hold areas are named Old Hold and New Hold. Each of them is preceded by an attribute (A) word containing the length of the particular hold area. These hold areas appear as follows:



Further discussion of the use of these hold areas by the control-level compare and extraction routines may be found under Control Level Processing.

Pseudo Registers

The first 16 words of the RPG object program are set aside for use as pseudo registers. They are used by many of the object time routines for passing addresses and other information. These registers serve the following functions:

Pseudo registers 0-1	- volatile
Pseudo register 2	- contains the address of the I/O buffer for the record being processed.
Pseudo registers 3-7	- volatile
Pseudo register 8	- contains parameters during the GET function.
Pseudo registers 9-13	- volatile
Pseudo register 14	- used to pass return addresses.
Pseudo register 15	- contains the address of the IOD being used.

OBJECT TIME ROUTINES

Since the object time tables and work areas have now been examined, a detailed description of the executable object program routines may now be presented.

Input/Output Drivers (IODs)

One IOD is generated for every file specified in the RPG source program. This IOD is a routine which provides linkage to a library subroutine that will perform the actual input or output operation for the file.

There are six card and printer IODs; they will precede disk IODs in the generated object program.

Called by IOD	Device
PRNT3	IBM 1403 Printer
PRNT1	IBM 1132 Printer
WRTY0	Console Printer
PNCH0	IBM 1442-5 Card Punch
CARD0	IBM 1442 Card Read Punch
READ0	IBM 2501 Card Reader

The logic of these IODs varies from device to device, but the card and printer IODs do have the first five words in common, and they contain the following information:

Word1 - address of the Write entry point.
Word2 - address of the Read entry point.
Word3 - address of the Control entry point.
Word4 - address of the I/O area.
Word5 - address of the Wait entry point.

If any of these addresses do not apply to a particular IOD (e.g., Word2 does not apply to a printer), the word will contain zeroes before being relocated by the Core Load Builder at load time.

As previously mentioned, disk IODs are generated immediately following the card and printer IODs. They provide linkage to library subroutines that perform actual I/O operations for disk files. The library subroutines which may be called by disk IODs are:

Type of File	Subroutine Name	Function
I. Sequential Files Processed Randomly	DAOPN DAIO DACLS	Open the file Read-write operations Close the file
II. Sequential Files Processed Sequentially	SEQOP SEQIO SEQCL	Open the file Read-write operations Close the file
III. Indexed-Sequential Files		
A. Load	ISLDO ISLD ISLDC	Open the file Load records Close the file
B. Add	ISADO ISAD ISADC	Open the file Add records Close the file
C. Sequential (Input or Output)	ISEQO ISEQ ISEQC	Open the file Read-write operations Close the file
D. Random	ISRDO ISRD ISRDC	Open the file Retrieve or update Close the file

Again, the logic of the disk IODs varies from one type of processing to another, but the first six words of any disk IOD contain the following information:

Word1 - address of the PUT entry to the IOD.
Word2 - address of the GET entry to the IOD.
Word3 - address of the OPEN entry to the IOD.
Word4 - address of the I/O area.
Word5 - address of the WAIT entry to the IOD.
Word6 - address of the CLOSE entry to the IOD.

Also, if any of these words do not apply to a particular IOD, the word will contain zeroes before being relocated by the Core Load Builder at load time.

Fixed Driver (Overhead)

The fixed driver routine functions as the main linkage driver for every RPG object program. It is always the same length and performs the same functions. Since this section of the object code does not lend itself to flowcharting, the actual code is shown here (Figure 13). Following the code, the functions of the more important labels (circled) will be examined in detail.

```
                    0162   **                                          Y5801620
                    0163   **     RPG      OBJECT TIME COMMUNICATION REGION      Y5801630
                    0164   **                                          Y5801640
                    0165   **                                          Y5801650
073E 0  0000        0166   R0     DC       *-*        PSEUDO REGISTER 0     Y5801660
073F 0  0000        0167   R1     DC       *-*        PSEUDO REGISTER 1     Y5801670
0740 0  0000        0168   R2     DC       *-*        PSEUDO REGISTER 2     Y5801680
0741 0  0000        0169   R3     DC       *-*        PSEUDO REGISTER 3     Y5801690
0742 0  0000        0170   R4     DC       *-*        PSEUDO REGISTER 4     Y5801700
0743 0  0000        0171   R5     DC       *-*        PSEUDO REGISTER 5     Y5801710
0744 0  0000        0172   R6     DC       *-*        PSEUDO REGISTER 6     Y5801720
0745 0  0000        0173   R7     DC       *-*        PSEUDO REGISTER 7     Y5801730
0746 0  0000        0174   R8     DC       *-*        PSEUDO REGISTER 8     Y5801740
0747 0  0000        0175   R9     DC       *-*        PSEUDO REGISTER 9     Y5801750
0748 0  0000        0176   R10    DC       *-*        PSEUDO REGISTER 10    Y5801760
0749 0  0000        0177   R11    DC       *-*        PSEUDO REGISTER 11    Y5801770
074A 0  0000        0178   R12    DC       *-*        PSEUDO REGISTER 12    Y5801780
074B 0  0000        0179   R13    DC       *-*        PSEUDO REGISTER 13    Y5801790
074C 0  0000        0180   R14    DC       *-*        PSEUDO REGISTER 14    Y5801800
074D 0  0000        0181   R15    DC       *-*        PSEUDO REGISTER 15    Y5801810
                    0182   *****                                       Y5801820
074E 00 4C000000    0183   CNTRL  BSC   L  *-*        CONTROL PASSER        Y5801830
                    0184   *****                                       Y5801840
0750 0  0000        0185   ADSRT  DC       *-*        STARTING ADDRESS OF OBJECT PY5801850
0751 0  0000        0186   TABOT  DC       *-*        ADDR OF TABLE OUTPUT ROUT   Y5801860
0752 0  0000        0187   HDAD   DC       *-*        HEADING AND DETAIL LINE ADDRY5801870
0753 0  0000        0188   HDTAB  DC       *-*        HEADING AND DETAIL LINE TABLY5801880
0754 0  0000        0189   TAD    DC       *-*        TOTAL LINE ADDR             Y5801890
0755 0  0000        0190   TTAB   DC       *-*        TOTAL LINE TABLE            Y5801900
0756 0  0000        0191   DCALC  DC       *-*        DETAIL CALC ADDR            Y5801910
0757 0  0000        0192   TCALC  DC       *-*        TOTAL CALC ADDR             Y5801920
0758 0  0000        0193   CHAN1  DC       *-*        ADDR OF CHAIN ROUT 1        Y5801930
0759 0  0000        0194   CHAN2  DC       *-*        ADDR OF CHAIN ROUT 2        Y5801940
075A 0  0000        0195   CHAN3  DC       *-*        ADDR OF CHAIN ROUT 3        Y5801950
075B 0  0000        0196   STERI  DC       *-*        ADDR OF STERLING INPUT      Y5801960
075C 0  0000        0197   RAFAD  DC       *-*        ADDR OF RAF FILE ROUR       Y5801970
075D 0  0000        0198   STERO  DC       *-*        ADDR OF STERLING OUTPUT     Y5801980
075E 0  0000        0199   RAFIO  DC       *-*        I/O ADDR OF TO FILE FOR RAF Y5801990
075F 0  0000        0200   TABLD  DC       *-*        ADDR OF TABLE LOAD ROUTINE  Y5802000
0760 0  0000        0201   CTLFD  DC       *-*        ADDR OF CONTROL FIELD ROUT  Y5802010
0761 0  0000        0202   LOWFD  DC       *-*        ADDR OF LOWFIELD            Y5802020
0762 0  0000        0203   CHSAV  DC       *-*        BACKUP FOR CHAIN1 ROUT ADDR Y5802030
0763 0  0000        0204   EAD    DC       *-*        EXCPT LINE ADDR             Y5802040
0764 0  0000        0205   RTE    DC       *-*        RETURN ADDR AFTER EXCPT LINEY5802050
0765 0  0000        0206   ETAB   DC       *-*        EXCPT LINE TABLE            Y5802060
0766 0  0000        0207   ENDAD  DC       *-*        ADDR OF CLOSE FILES         Y5802070
0767    0005        0208          BSS      5          FOR FUTURE EXPANSION        Y5802080
076C 00 7401000C    0209   INPSE  MDM      R11-AF,ONEE    BUMP R11 BY 1           Y5802090
076E 00 7403000C    0210   ALPSE  MDM      R11-AF,THREE   BUMP R11 BY 3           Y5802100
0770 00 C480000C    0211   TSTRC  LD    I  R11-AF     GET FIRST WD OF FILE TAB    Y5802110
0772 0  D0CC        0212          STO      R1         SAVE IN R1                  Y5802120
0773 00 4C800002    0213          B     I  R1-AF      GO TO ADDR IN R1            Y5802130
0775 0              0214   ERR    EQU      *          PUT RCD OUT OF SEQUENCE     Y5802140
0775 00 65800010    0215          LDX   I1 R15-AF     SET XR1 TO R15              Y5802150
0777 0  C00C        0216          LD       X0000      GET A 0                     Y5802160
0778 0  D101        0217          STO    1 ONEE       SET SWITCH IN NUM SEQ RTN   Y5802170
0779 00 C400012B    0218          LD    L  ACCST-AF   SEE IF R15+1 WAS /F0        Y5802180
077B 00 4CA00010    0219          BNZ   I  R15-AF     YES TA ADDR IN R15          Y5802190
077D 0  C007        0220          LD       X7009      BRANCH TO *+9               Y5802200
077E 0  D107        0221          STO    1 SEVEN      SET SWITCH IN NUM SEQ RT    Y5802210
077F 30 191C5659    0222          CALL     RGERR      GET OBJECT TIME ERROR ROUT  Y5802220
0781 0  1111        0223          DC       /1111      NOTE C111                   Y5802230
0782 00 4C000049    0224          B     L  GETRC-AF   GO READ A RECORD            Y5802240
0784 0  0000        0225   X0000  DC       /0000      THESE TWO DC'S SET SWITCH   Y5802250
0785 0  7009        0226   X7009  DC       /7009      IN THE NUM SEQ RTN          Y5802260
```

Figure 13. Object Code for the Fixed Driver Routine (Part 1 of 4)

SOURCE DATE-02/25/69

```
ADDR REL OBJECT    ST.NO.  LABEL OPCD FT OPERANDS                                        ID/SEQNO

0786 0             0227    GETRC EQU      *                                              Y5802270
0786 00 C400015F   0228    HLTSW LD   L   H1-1-AF       GET LR INDICATOR                 Y5802280
0788 C0 4C2000F4   0229          BNZ      EOJRO-AF      IF ON GO TO EOJ                  Y5802290
078A 00 65000160   0230          LDX  L1  H1-AF         POINT TO START OF HALT IND       Y5802300
078C 0  C01C       0231          LD       NHIND         HOW MANY ARE THERE TO CHK        Y5802310
078D 0  D01C       0232          STO      LHIND         STORE FOR LOOP CONTROL           Y5802320
078E 0  C100       0233    LOOP  LD    1  ZEROE         GET A HALT INDICATOR             Y5802330
078F 00 4C2000C4   0234          BNZ      HLTMS-AF      IF ON GO PUT OUT MESSAGE         Y5802340
0791 0  7101       0235          MDX   1  ONEE          POINT TO NEXT HALT IND           Y5802350
0792 00 74FF006D   0236          MDM      LHIND-AF,-1  DECREMENT LOOP COUNTER            Y5802360
0794 C  70F9       0237          B        LOOP          IF ALL NOT CHECKED RETRY         Y5802370
                   0238    *                                                             Y5802380
0795 0             0239    RESRT EQU      *                                              Y5802390
                   0240    * ALL CONTROL LEVEL,HALT,AND INPUT RECORD INDICATOR           Y5802400
                   0241    * SWITCHES ARE NOW TURNED OFF                                 Y5802410
0795 00 65000154   0242          LDX  L1  FP-AF         POINT TO START OF INDICATOR      Y5802420
                   0243    * TO SHUT OFF                                                 Y5802430
0797 0  C013       0244          LD       NUMIN         HOW MANY TO SHUT OFF             Y5802440
0798 0  D013       0245          STO      NUMLP         STORE FOR LOOP CONTROL           Y5802450
0799 0  1010       0246          SLA      SIXTE         CLEAR THE ACCUMULATOR            Y5802460
079A 0  D100       0247    LOOP1 STO   1  ZEROE         SHUT OFF AN INDICATOR            Y5802470
079B 0  7101       0248          MDX   1  ONEE          POINT TO NEXT INDICATOR          Y5802480
079C 00 74FF006F   0249          MDM      NUMLP-AF,-1  DECREMENT LOOP COUNTER            Y5802490
079E 0  70FB       0250          B        LOOP1         IF ALL NOT OFF CONTINUE          Y5802500
079F 0  C0C1       0251          LD       LOWFD         GET ADDRESS OF LOW FIELD         Y5802510
07A0 0  D0A7       0252          STO      R10           PUT IN REG 10                    Y5802520
07A1 00 C480000B   0253          LD    I  R10-AF        GET ADDR OF GET                  Y5802530
07A3 0  D0A2       0254          STO      R8            PUT IN REG 8                     Y5802540
07A4 0  C008       0255          LD       INDON         GET INDICATOR ON                 Y5802550
07A5 00 D4000155   0256          STO   L  L0-AF         TURN LEVEL ZERO ON               Y5802560
07A7 00 4C800009   0257    LABEO B     I  R8-AF         GO TO CONTENTS OF REG 8          Y5802570
07A9 C  0009       0258    NHIND DC       9             NUMBER OF HALT INDICATORS        Y5802580
07AA 0  0000       0259    LHIND DC       0             LOOP COUNTER                     Y5802590
07AB 0  0000       0260    NUMIN DC       0             FILLED IN AT COMPILE TIME        Y5802600
07AC 0  0000       0261    NUMLP DC       0             LOOP COUNTER                     Y5802610
07AD 0  0001       0262    INDON DC       /0C01         INDICATOR ON                     Y5802620
07AE 00 C400015F   0263    GETIF LD    L  LR-AF         GET LR INDICATOR                 Y5802630
07B0 0  90FC       0264          S        INDON         IS IT ON                         Y5802640
07B1 00 4C180103   0265          BZ       CLOSE-AF      YES END OF FILE                  Y5802650
07B3 00 65800024   0266          LDX  I1  LOWFD-AF      GET ADDR OF LOW FIELD            Y5802660
07B5 00 6D000002   0267          STX  L1  R1-AF         SAVE IN REG 1                    Y5802670
07B7 0  C102       0268          LD    1  TWOE          ADDR OF MOVE FIELDS RTN          Y5802680
07B8 00 D4000009   0269          STO   L  R8-AF         SAVE IN REG 8                    Y5802690
07BA 0  C100       0270    LDIOA LD    1  ZEROE         GET LOW FIELD                    Y5802700
07BB 00 D4000003   0271          STO   L  R2-AF         SAVE IN REG 2                    Y5802710
07BD 00 65800003   0272          LDX  I1  R2-AF         PICK UP ADDR OF GET              Y5802720
07BF 0  C106       0273          LD    1  SIXE          GET IOD ADDRESS                  Y5802730
07C0 00 D4000003   0274          STO   L  R2-AF         SAVE IN REG 2                    Y5802740
07C2 00 65800003   0275          LDX  I1  R2-AF         PUT IT IN XR1                    Y5802750
07C4 0  C103       0276          LD    1  THREE         GET ADDR THE I/O AREA            Y5802760
07C5 00 D4000003   0277          STO   L  R2-AF         PUT IN REG 2                     Y5802770
07C7 00 C400014F   0278          LD    L  INTMR-AF      TURN ON MR SWITCH IF NECC        Y5802780
07C9 0  180F       0279          SRA      15            MAKE EQUAL TO /0001              Y5802790
07CA 00 D4000150   0280          STO   L  MR-AF         ESSARY                           Y5802800
07CC 0  1810       0281          SRA      SIXTE         CLEAR ACCUMULATOR                Y5802810
07CD 00 D400014F   0282          STO   L  INTMR-AF      SHUT OFF INTERNAL MR SWITC       Y5802820
07CF 0  70D7       0283          B        LABEO         GO TO MOVE INPUT FIELDS          Y5802830
                   0284    *                                                             Y5802840
                   0285    *                                                             Y5802850
```

Figure 13. Object Code for the Fixed Driver Routine (Part 2 of 4)

```
                                                                     SOURCE DATE-02/25/69

ADDR REL OBJECT    ST.NO.  LABEL OPCD FT OPERANDS                              ID/SEQNO

07D0 0             0286    PRORC EQU      *                                    Y5802860
07D0 00 6580000B   0287          LDX  I1 R10-AF    GET CONTENTS OF REG 10      Y5802870
07D2 0  C103       0288          LD    1 THREE     GET INPUT INDICATOR ADDR    Y5802880
07D3 0  D002       0289          STO     SETIN+2                               Y5802890
07D4 0  CCD8       0290    SETIN LD      INDON     GET INDICATOR ON            Y5802900
07D5 00 D4000000   0291          STO  L  ZFROE     THIS TURNS ON INPUT IND     Y5802910
07D7 00 C4000004   0292          LD   L  R3-AF     GET REG 3                   Y5802920
07D9 00 94000010   0293          S    L  R15-AF    IS IT EQUAL TO REG 15       Y5802930
07DB 00 4C180119   0294          BZ      TOTSW-AF  YES TO TOTAL TIME ROUTINE   Y5802940
07DD 00 6580000B   0295          LDX  I1 R10-AF    POINT TO LCW FIELD          Y5802950
07DF 0  C101       0296          LD    1 ONEE      ADDR OF LEVEL EXTRACTION    Y5802960
07E0 00 D4000009   0297          STO  L  R8-AF     PUT IT IN REG 8             Y5802970
07E2 0  70C4       0298          B       LABEO     GO TO LABEL                 Y5802980
                   0299    *THIS ROUTINE IS TO GET A FILE                      Y5802990
07E3 0             0300    ADDGT EQU     *                                     Y5803000
07E3 00 65800009   0301          LDX  I1 R8-AF     GET REG 8                   Y5803010
07E5 0  C101       0302          LD    1 ONEE      GET FILTAB-4 ADDR           Y5803020
07E6 00 D400000C   0303          STO  L  R11-AF    PUT IT IN REG 11            Y5803030
07E8 00 6D00000F   0304          STX  L1 R14-AF    PUT REG 8 IN REG 14         Y5803040
07EA 0  C100       0305    GETFL LD    1 ZEROE     GET IOD ADDRESS             Y5803050
07EB 00 D4000010   0306          STO  L  R15-AF    STORE IN REG 15             Y5803060
07ED 00 65800010   0307          LDX  I1 R15-AF    GET ADDR IN REG 15          Y5803070
07EF 0  C103       0308          LD    1 THREE     GET I/O AREA ADDRESS        Y5803080
07F0 00 D4000003   0309          STO  L  R2-AF     SAVE IN REG 2               Y5803090
07F2 0  C101       0310          LD    1 ONEE      GET READ ENTRY ADDRESS      Y5803100
07F3 00 D4000010   0311          STO  L  R15-AF    SAVE IN REG 15              Y5803110
07F5 0  C004       0312          LD      AINPS     GET ADDR OF INPSE           Y5803120
07F6 00 D400000B   0313          STO  L  R10-AF    SAVE IN REG 10              Y5803130
07F8 00 4C800010   0314          B    I  R15-AF    GO TO READ ENTRY IN IOD     Y5803140
07FA 0  002F       0315    AINPS DC      INPSE-AF  ADDR OF INPSE               Y5803150
                   0316    *INPUT ROUTINE LINKAGE TO ALPHA SEQUENCE            Y5803160
                   0317    *MULTIFIELD LINKAGE ROUTINE                         Y5803170
07FB 0             0318    MFLNK EQU     *         RTN TO HANDLE MULTI-FILE    Y5803180
07FB 00 6580000C   0319          LDX  I1 R11-AF    POINT TO FILE INPUT TABL    Y5803190
07FD 0  C101       0320          LD    1 ONEE      GET MFEXT RTN ADDR          Y5803200
07FE 00 D4000009   0321          STO  L  R8-AF     SAVE IN REG 8               Y5803210
0800 0  70A6       0322          B       LABEO     GO TO LABEL                 Y5803220
                   0323    * ROUTINE TO PRINT HALT MESSAGE INDICATOR           Y5803230
0801 0             0324    HLTMS EQU     *                                     Y5803240
0801 0  C01D       0325          LD      AH1M1     GET ADDR OF H1-1            Y5803250
0802 00 D400000F   0326          STO  L  R14-AF    PUT IN REG 14               Y5803260
0804 0  C015       0327          LD      ZERO      GET COMPARAND               Y5803270
0805 00 D4000010   0328          STO  L  R15-AF    SAVE IN REG 15              Y5803280
0807 00 7401000F   0329    ITERA MDM     R14-AF,ONEE  BUMP REG 14 BY 1         Y5803290
0809 00 74010010   0330          MDM     R15-AF,ONEE    BUMP REG 15 BY 1       Y5803300
080B 00 C480000F   0331          LD   I  R14-AF    GET A HALT INDICATOR        Y5803310
080D 00 94000070   0332          S    L  INDON-AF  IS IT ON                    Y5803320
080F 00 4C2000CA   0333          BNZ     ITERA-AF  NO TRY NEXT ONE             Y5803330
0811 0  C00C       0334          LD      NOTM1     GET MESSAGE TO PUT OUT      Y5803340
0812 00 EC000010   0335          OR   L  R15-AF    OR IN HALT NUMBER           Y5803350
0814 0  D002       0336          STO     NOTMS     UPDATE MESSAGE              Y5803360
0815 30 191C5659   0337          CALL    RGERR     GET OBJECT TIME ERROR ROUT  Y5803370
0817 0  1120       0338    NOTMS DC      /1120     C12N MESSAGE, N = HALT NUMB Y5803380
0818 00 4C000058   0339          B    L  RESRT-AF  CONTINUE PROCESSING         Y5803390
081A 0  0000       0340    ZERO  DC      0         CONSTANT OF ZERO            Y5803400
081B 0  0000       0341    SEEKA DC      *-*       GENERAL WORK AREA           Y5803410
081C 0  0000       0342          DC      *-*       GENERAL WORK AREA           Y5803420
081D 0  0000       0343          DC      *-*       GENERAL WORK AREA           Y5803430
081E 0  1120       0344    NOTM1 DC      /112C     USED TO BUILD C12N NOTE     Y5803440
```

Figure 13. Object Code for the Fixed Driver Routine (Part 3 of 4)

```
                                                                     SOURCE DATE-02/25/69

ADDR REL OBJECT    ST.NO.  LABEL OPCD FT OPERANDS                                   ID/SEQNO

081F 0  015F        0345   AH1M1 DC      H1-1-AF     ADDRESS OF H1-1               Y5803450
0820 0              0346   TCLNK EQU     *           TOTAL TIME ROUTINE            Y5803460
0820 0  C00F        0347         LD      CON         HAS CON BEEN FILLED IN        Y5803470
0821 00 4C1800ED    0348         BZ      TOTRO-AF    NO TO TOTAL TIME              Y5803480
0823 00 65800010    0349         LDX  I1 R15-AF      GET ADDR IN REG 15            Y5803490
0825 0  C104        0350         LD    1 FOURE       GET R15+4                     Y5803500
0826 0  E009        0351         AND     CON         FIRST TIME SWITCH             Y5803510
0827 0  9008        0352         S       CON                                       Y5803520
0828 00 4C180071    0353         BZ      GETIF-AF    IF OK TO GETIF                Y5803530
082A 00 C400001A    0354   TOTRO LD   L  TCALC-AF    GET ADDR OF TOTAL CALCS       Y5803540
082C 00 D4000009    0355         STO  L  R8-AF       PUT IN REG 8                  Y5803550
082E 00 4C800009    0356         B    I  R8-AF       GO TO TOTAL CALCS             Y5803560
0830 0  0000        0357   CON   DC      *-*         FILLED IN EXTERNALLY          Y5803570
                    0358   **END OF JOB ROUTINE                                    Y5803580
0831 0              0359   EOJRO EQU     *           END JOB ENTRY                 Y5803590
0831 0              0360   EOJBK EQU     *           END JOB ENTRY                 Y5803600
0831 00 65000155    0361         LDX  L1 LO-AF       POINT TO START OF LEVEL       Y5803610
                    0362   * INDICATORS TO MAKE FINAL CONTROL BREAK                Y5803620
0833 0  C100        0363         LD    1 ZEROE       PICK UP LO (CN)               Y5803630
0834 0  D101        0364   LOOP2 STO   1 ONEE        TURN ON AN INDICATOR          Y5803640
0835 0  7101        0365         MDX   1 ONEE        POINT TO NEXT INDICATOR       Y5803650
0836 00 74FF0102    0366         MDM     NLIND-AF,-1  DECREMENT LOP COUNTER        Y5803660
0838 0  70FB        0367         B       LOOP2       IF NOT DONE SET NEXT ONE      Y5803670
0839 00 C400001A    0368         LD   L  TCALC-AF    GET TOTAL CALCULATION ADDR    Y5803680
083B 00 D4000009    0369         STO  L  R8-AF       SAVE IN REG 8                 Y5803690
083D 00 4C800009    0370         B    I  R8-AF       GO THERE                      Y5803700
083F 0  000A        0371   NLIND DC      10          L1-L9,LR                      Y5803710
                    0372   **CLOSE THE FILES                                       Y5803720
0840 00 C4000014    0373   CLOSE LD   L  TABOT-AF    GET ADDR OF TABLE OUTPUT      Y5803730
0842 00 D4000005    0374         STO  L  R4-AF       PUT IN R4                     Y5803740
0844 0  1010        0375         SLA     16          CLEAR THE ACCUMULATOR         Y5803750
0845 00 D4000007    0376         STO  L  R6-AF       CLEAR R6                      Y5803760
0847 00 D4000008    0377         STO  L  R7-AF       R7                            Y5803770
0849 00 65000112    0378         LDX  L1 *+4-AF      SET UP RETURN ADDRESS         Y5803780
084B 00 6D00000A    0379         STX  L1 R9-AF       SAVE IN REG 9                 Y5803790
084D 00 4C800005    0380         B    I  R4-AF       GO CLEAN UP                   Y5803800
084F 00 4C800029    0381   CL1   B    I  ENDAD-AF    GO CLOSE FILES                Y5803810
                    0382   ***                                                     Y5803820
0851 0  6038        0383   EOJ   EXIT                ALL DONE                      Y5803830
                    0384   ***                                                     Y5803840
0852 0  0000        0385   FC1   DC      *-*         CHAINING INDICATOR 1          Y5803850
0853 0  0000        0386   FC2   DC      *-*         CHAINING INDICATOR 2          Y5803860
0854 0  0000        0387   FC3   DC      *-*         CHAINING INDICATOR 3          Y5803870
0855 0  0000        0388   C1FTB DC      0           ADDR OF CHAIN1 FILE TABLE     Y5803880
0856 0  7001        0389   TOTSW B       *+1         FIRST TIME DONT GO TO TOTR    Y5803890
0857 0  7002        0390         B       TOTRO       AFTER 1ST TIME GO TO TOTRO    Y5803900
0858 0  C003        0391         LD      NOOP        SET AFTER 1ST TIME BRANCH     Y5803910
0859 0  D0FC        0392         STO     TOTSW       TO GO TO TOTRO                Y5803920
085A 00 4C000071    0393         B    L  GETIF-AF    1ST TIME ONLY GO TO GETIF     Y5803930
085C 0  1000        0394   NOOP  NOP                 CONSTANT FOR NOP              Y5803940
                    0395   *                                                       Y5803950
```

Figure 13. Object Code for the Fixed Driver Routine (Part 4 of 4)

INPSE: This code is entered from the GET routine for a particular file after a record has been read from that file. PR11 (pseudo register 11) contains the address of the File Input Table -4 for the file just read. (See File Input Tables for further information.) PR11 is then incremented by 4 to point at the first word of the table entry. This word (address of the INPR routine) is then placed in PR1 and the Fixed Driver branches to the INPR routine.

ALPSE: This code is entered from the output lines routine. PR11 contains the address of the Output Table -3. The Output Table in question may be DTAB, TOTAB, OTAB, or EXTAB. PR11 is then incremented by 3 to point at Word1 of the table entry. Word1 (address of the Test Output Indicators routine) is then placed in PR1 and the branch is taken to the Test Output Indicators routine.

ERR: If a numeric sequencing error occurs, this routine is entered to put out the error indication.

GETRC: This routine first checks the LR indicator; if on, a branch is taken to total calculations. If off, the halt indicators are checked next. If any of the halt indicators are on, a branch is taken to HLTMS, the routine which indicates the error. Next, beginning at RESRT, all control-level, halt, and input record indicators are turned off. The address of the GET routine for the file about to be read is taken from Word1 of Low Field and placed in PR8. A branch to the GET routine is then taken via PR8.

GETIF: This routine first checks the LR indicator; if on, a branch is taken to the CLOSE routine. If off, the address of the move input fields routine is taken from Word3 of Low Field and placed in PR8. The IOD address is obtained and from the IOD the I/O area address (Word4) is taken and placed in PR2. The MR indicator is set on/off and a branch is taken to LABEO in the Fixed Driver. At LABEO a branch via PR8 is taken to the move input fields (INPF) routine for the record type being processed.

PRORC: This code takes the record identifying indicator address from Word4 of Low Field and turns that indicator on. If there are control levels in the program, a branch is taken to the control-level extraction routine pointed at by Word2 of Low Field. If no control levels exists in the program, a branch is taken to total calculations, providing this is not the first cycle of the object program.

ADDGT: Upon entering this code, PR8 is pointing at two words in the GET routine for the particular file from which a record is about to be retrieved. The first word contains the address of the IOD for the file; the second word contains the address of the File Input Table entry -4. The File Input Table -4 address is placed in PR11. PR14 is loaded with the contents of PR8. At GETFL, the I/O area address is taken from the IOD and placed in PR2 and the read entry address is taken from the IOD and placed in PR15. The address of INPSE in the Fixed Driver is placed in PR10 and a branch is taken to the read entry point in the IOD to read a record.

MFLNK: This code provides linkage to handle matching fields extraction and multifile logic. It is entered from a GET routine with PR11 pointing to the File Input Table entry which corresponds to the record type just read. Word2 of the entry (address of the matching fields extraction routine for this record type) is taken from the table and a branch is taken to this address (MFEXT).

HLTMS: This code checks the halt indicators and provides linkage to issue any necessary error indication.

TCLNK: This code tests a first-time switch to determine whether total calculations and lines should be bypassed. If this is the first cycle of the object program, a branch is taken to GETIF in the Fixed Driver. If not, a branch is taken to total calculations.

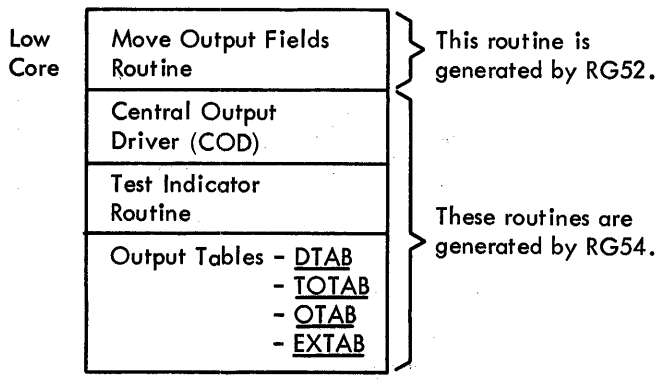
EOJRO: This code turns LR and L1-L9 on and links to total calculations.

CLOSE: This code obtains the address of the table output routine, places it in PR4 and links to the Close Files routine.

EOJ: Exit.

Output Lines Routines

The object code routines for putting out lines consist of four major sections, not including the IODs which are essential to the operation of these routines. These routines appear in core storage in the following order:



See Figure 17 for the core storage locations of all object routines.

Each of these routines will be discussed separately, after which their combined function will be explained.

Move Output Fields Routine

For each output line specified in the RPG source program, a move output fields routine will be generated. This routine moves the fields which make up its particular line into the output buffer (I/O area). If field output is conditioned by an indicator, the routine ensures that the indicator is on/off before the move is performed.

Central Output Driver (COD)

The COD first receives control whenever output is to be performed. The accompanying figure describes the logic of the COD. Note that this routine has several entry points and several exits. The COD is a pre-coded routine in RG54 and is generated for every RPG object program.

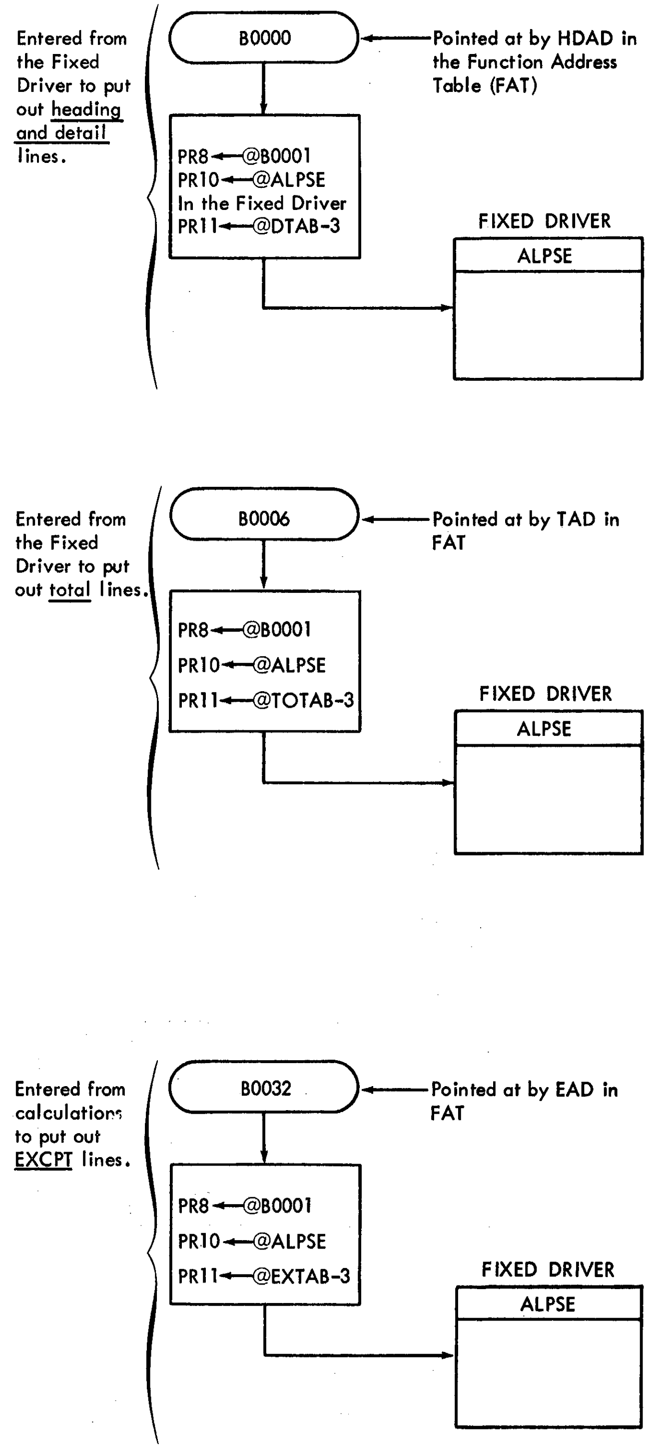


Figure 14. Logic of the Central Output Driver (Part 1 of 3)

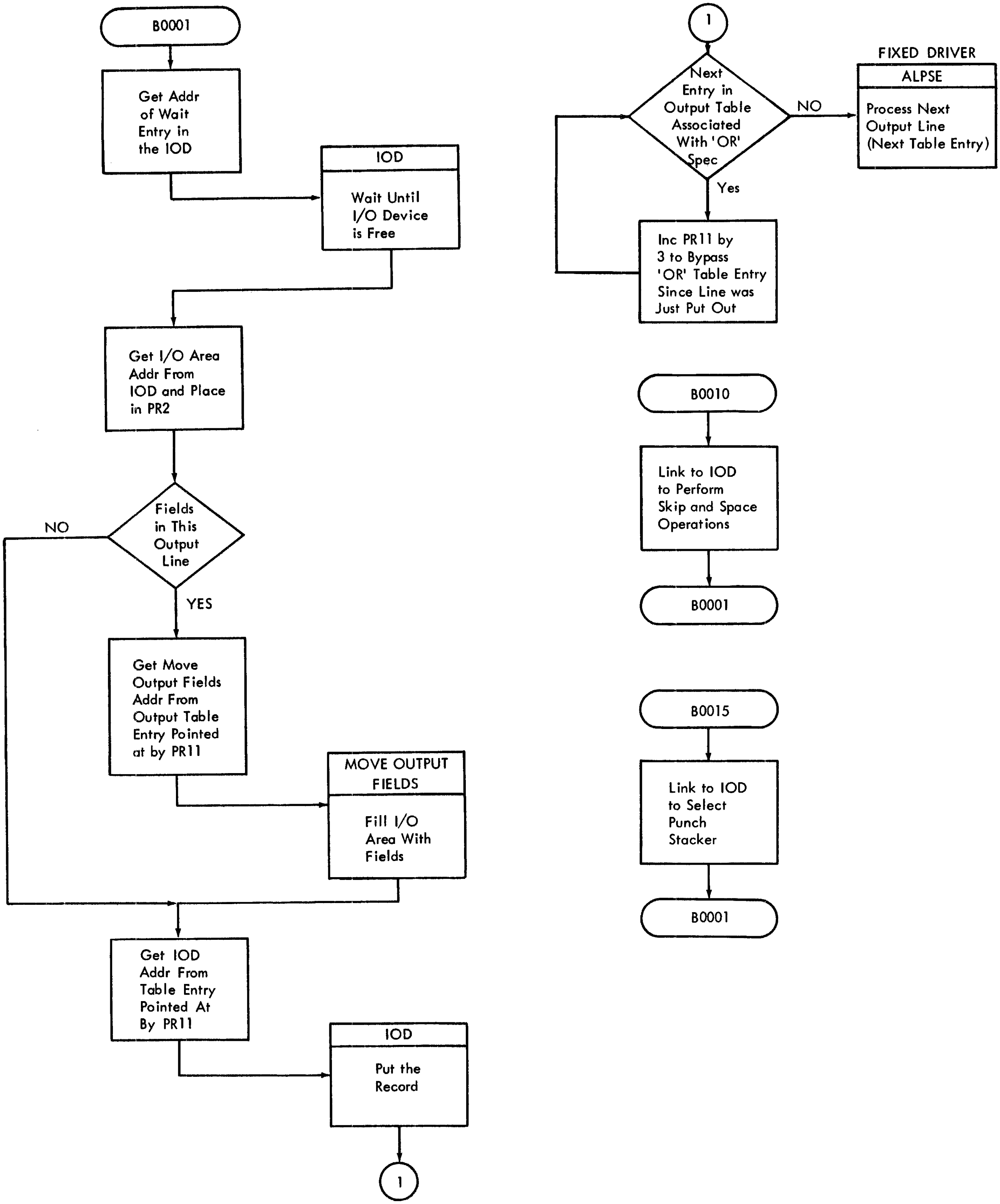


Figure 14. Logic of the Central Output Driver (Part 2 of 3)

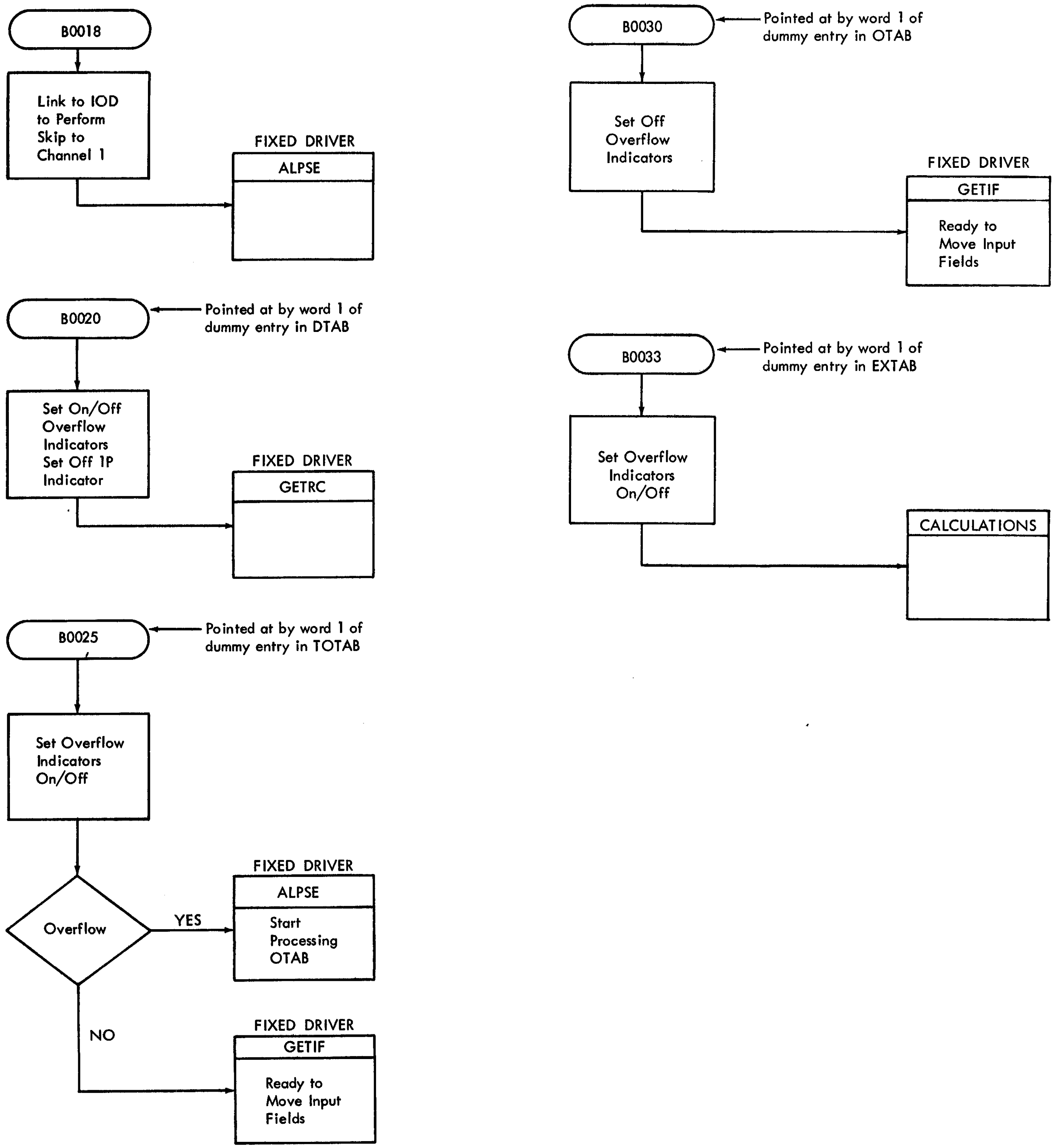
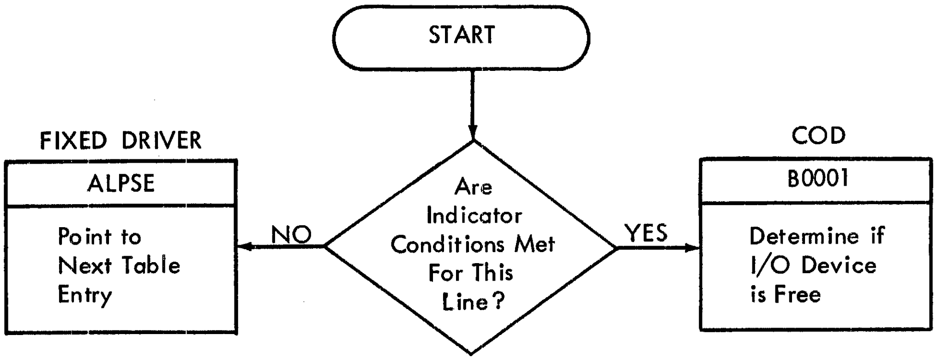


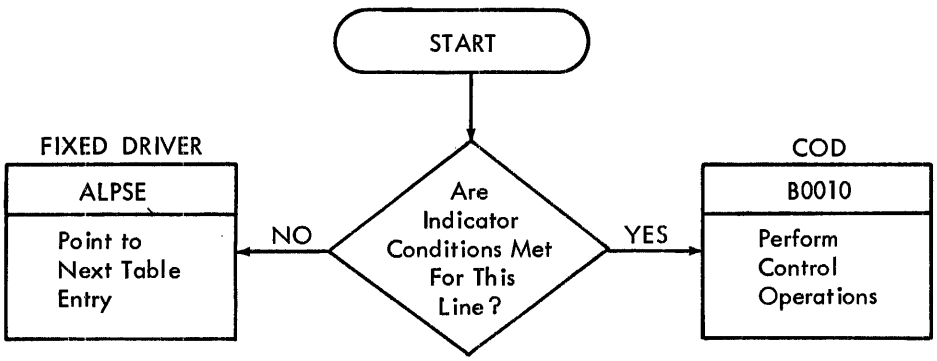
Figure 14. Logic of the Central Output Driver (Part 3 of 3)

Test Indicators Routines

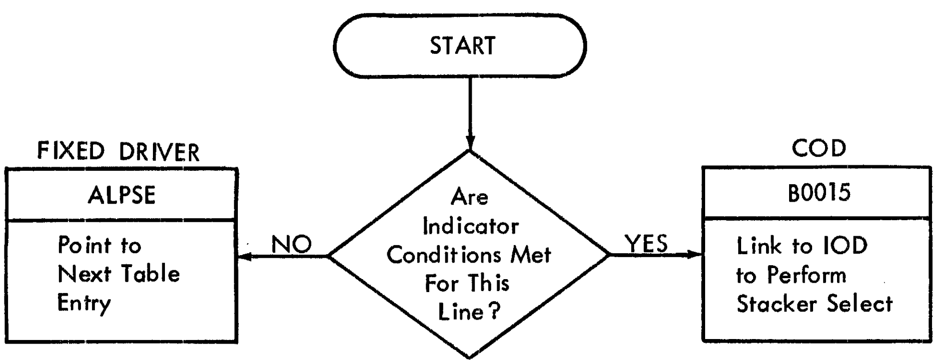
For every output line specified in the source program, a test indicators routine is generated. For most output line specifications the following routine will be generated.



If the output line is directed to a printer and space before or skip before is specified, the following routine will be generated.



If the output line is directed to a punch and stacker select is specified, the following test indicators routine will be generated.



Output Tables

A description of these tables is contained under Tables and Work Areas. These tables follow the Test Indicators routines in core storage and are contiguous in the following order.

Low Core
DTAB
TOTAB
OTAB
EXTAB

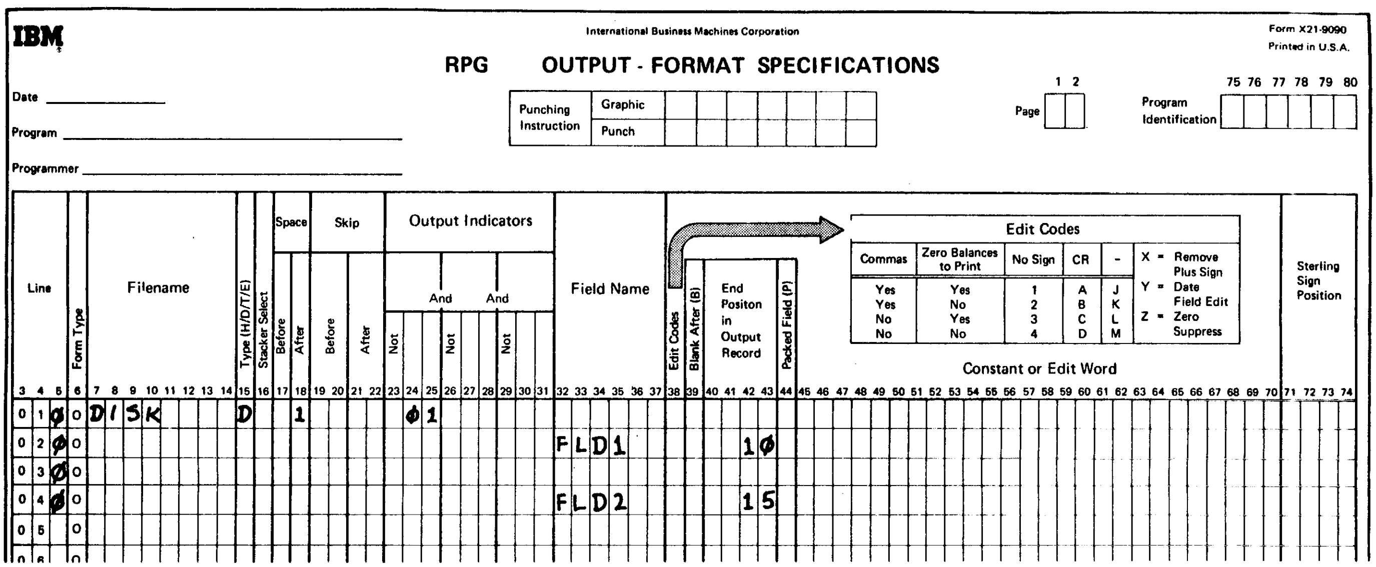
Starting with this basic information about the functions and logic of the output lines routines, the following section shall attempt to show the dependencies and interrelationships of these routines within an actual object program.

To illustrate these relationships, the output specifications of an RPG source program are provided so that a trace of the code may be made.

A detail lines output table (DTAB) will be generated with these entries:

DTAB		
Address of the routine that tests indicator 01	Address of the routine that moves FLD1 to the output buffer	Address of the IOD that performs the I/O operation
Address of the routine that tests indicator 02	Address of the routine that moves FLD2 to the output buffer	Address of the IOD that performs this I/O operation
Address of B0020 in the COD	Not used	Not used

Steps 1 through 9 trace the sequence of steps that will be executed at detail lines time.



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date, Program, Programmer, Punching Instruction (Graphic, Punch), Page, Program Identification

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Output Indicators	Field Name	End Position in Output Record
01Ø	O	DISK	D	1	Ø1		
02Ø	O					FLD1	1Ø
03Ø	O						
04Ø	O					FLD2	15
05	O						

1. The Central Output Driver (COD) receives control from the Fixed Driver at B0000. The address of B0001 is placed in PR8, and the address of ALPSE in the Fixed Driver is placed in PR10. The address of DTAB-3 is placed in PR11 and a branch is taken to ALPSE via PR10.
2. At ALPSE (this label is discussed in further detail under Fixed Driver), PR11 is incremented by 3 to point at Word1 of the first entry in DTAB. The address of the test indicators routine for the line associated with the first entry is taken from DTAB and placed in PR1. A branch is then taken to the test indicators routine.
3. The test indicators routine determines whether indicator 01 is on. If 01 were not on, a branch would be taken via PR10 to ALPSE which would increment PR11 and load PR1 to process the output line. Assuming though, that 01 is on, a branch is taken to the address contained in PR8. (PR8 was loaded with the address of B0001 in the COD.)
4. At B0001 the IOD address is taken from the table entry associated with the line being processed and a branch is taken to the WAIT entry point in the IOD. The IOD returns directly to the COD when the I/O device is free.
5. The COD then takes the address of the move output fields routine from the DTAB entry for the line being processed and branches to that routine.
6. The move output fields routine moves FLD1 to its proper place in the output buffer and links back to the COD.
7. The COD links to the PUT entry point of the IOD to put out the output line just built. Control is then given back to the COD.
8. Next, the move output fields routine addresses of the first and second entries in DTAB are compared. If equal, the COD would determine that entry2 is an 'OR' table entry. The 'OR' entry would not be processed since the line has already been put out. In the case of these lines however, the addresses are not equal and a branch is taken to ALPSE in the Fixed Driver.
9. At ALPSE, PR11, which still points to entry1 in DTAB, is incremented by 3 to point to entry2. The address of the test indicators routine for the second output line is taken and a branch is made to the test indicators routine.

Steps 3-9 are then repeated in order to put out the second output line. When step 9 is again reached and PR11 is incremented by 3, it will be pointing at the dummy table entry. Word1 of this entry will contain the address of B0020 in the COD and a branch will be taken to that point.

Next, if overflow occurred while printing the previous two lines, the appropriate overflow indicator is turned on. The 1P indicator is then turned off and a branch is taken to GETRC in the Fixed Driver. GETRC will begin the steps necessary to get the next record and repeat the cycle.

Get Input Record Routines

Certain tables and work areas essential to the task of getting an input record, namely, File Input Tables, IODs, Low Field, and the Low Field Block, PS, and Primary and Secondary Processing Blocks have been discussed previously under Tables and Work Areas. The reader should refer to that section of the publication for any information needed to understand the Get Input Record Routines.

These routines: GET routines,
MFTST routine,
EOFTS routine, and
MFEXT routines
will be explained individually and then a trace of a simple program will be made to illustrate how these routines function together to perform the entire "get record" task.

The GET Routines

One GET routine is generated for each primary or secondary input file specified in the source program. Following is the actual object code of each GET routine.

This routine is entered at the instruction labeled GET. The address of DTFA is then placed in PR8. (DTFA is a two-word area. The first word contains the address of the IOD for the input file associated with this GET routine, and the second word contains the address of the File Input Table -4 for this file.) A branch is then taken to ADDGT in the Fixed Driver (to GET the input record).

The GET routine is again entered at the instruction labeled RETUR. The address of the instruction LADD is placed in PR8 and a branch is taken via PR10 to INPSE in the Fixed Driver (to identify the record type just read).

The GET routine receives control again at LADD. The address of Low Field is placed in PR10 and, at NEWI, index register 2 is loaded with:

1. The address of Low Field if there are no secondary files in the program, or
2. The address of the primary processing block (PPB) if there are secondary files

```
                       0753   *                                                         Y4207530
0ABF 00 66000AC5       0754   GET    LDX  L2 DTFA       PUT RETURN                      Y4207540
0AC1 00 6E000009       0755          STX  L2 R8         ADDRESS IN R8                   Y4207550
0AC3 C0 4C0000A6       0756          B    L  ADGET      LINK TO OVERHEAD                Y4207560
0AC5 0  0000           0757   DTFA   DC      0          FILLED IN WITH DTF ADDR         Y4207570
0AC6 0  0000           0758          DC      0          FILLED IN WITH FILTAB-4 ADDRY4207580
0AC7 00 66000ACD       0759          LDX  L2 LADD       PUT RETURN                      Y4207590
0AC9 00 6E000009       0760          STX  L2 R8         ADDRESS IN R8                   Y4207600
0ACB 00 4C80000B       0761          B    I  R10        RETURN                          Y4207610
0ACD 00 66800024       0762   LADD   LDX  I2 DBLOW      GET LOWFLD ADDR                 Y4207620
0ACF 00 6E00000B       0763          STX  L2 R10                                        Y4207630
0AD1 00 66000000       0764   NEWI   LDX  L2 *-*        LOWFLD OR BUFFER ADDRESS        Y4207640
0AD3 00 6580000C       0765          LDX  I1 R11        GET ADDR FILTAB                 Y4207650
0AD5 0  C102           0766          LD    1 2          PICK UP CONTROL LEV ADDR        Y4207660
0AD6 0  D201           0767          STO   2 1          STORE IN 2ND LOWFLD SLOT        Y4207670
0AD7 0  C103           0768          LD    1 3          PICK UP INPF ADDR               Y4207680
0AD8 0  D202           0769          STO   2 2          STORE IN 3RD LOWFLD SLOT        Y4207690
0AD9 00 C4800009       0770          LD   I  R8         PICK UP RI ADDR                 Y4207700
0ADB 0  D203           0771          STO   2 3          PUT IN 4TH SLOT OF LOWFLD       Y4207710
0ADC 0  C006           0772   LEOF   LD      EOFWD                                      Y4207720
0ADD 0  EA03           0773          OR    2 3          OR WITH RI ADDR                 Y4207730
0ADE 0  D203           0774          STO   2 3          REPLACE                         Y4207740
0ADF 00 4C280000       0775   I34    BN      0          BRANCH IF EOF                   Y4207750
0AE1 00 4C0000BE       0776          B    L  MFLNK      GO TO OVERHEAD                  Y4207760
0AE3 0  0000           0777   EOFWD  DC      /0000      EOF MASK                        Y4207770
                       0778   *                                                         Y4207780
```

Figure 15. Object Code of the GET Routine

and this is the GET routine for the primary file, or

3. The address of the first secondary processing block (S1PB) if this is the GET routine for the 1st secondary file, or, the address of S2PB if this is the GET routine for the 2nd secondary file, etc.

At this time, PR11 is pointing to the File Input Table entry associated with the record type just identified. Word3 of the table entry (address of the control level extraction routine) is placed in the second word of the processing block or Low Field, whichever is pointed at by index register 2. Word4 (address of INPF) of the table entry is placed in Word3 of the processing block or Low Field. Next, the address of the record identifying indicator is placed in Word4 of the processing block (or Low Field) pointed to by index register 2.

At this time, if EOF had been encountered when the IOD had attempted to read the record, the high order bit of EOFWD would have been set to one. If EOF had not occurred, the bit would still be zero.

EOFWD is then ORed with the record identifying indicator address (Word4 of the block pointed to by index register 2). If EOF did occur, a branch is taken to the EOFTS routine unless there are secondary files in the program. If secondary files are present, the branch is taken to the MFTST routine. If EOF did not occur, a branch is taken to MFLNK in the Fixed Driver.

The MFTST Routine

The matching fields test (MFTST) routine is generated only when secondary files exist in the source program. The function of MFTST is to compare the matching fields of the primary and secondary files in order to select the proper record for processing and to determine the status of the matching record (MR) indicator.

Another function is to handle all EOF conditions on primary and secondary files.

This routine is pre-coded and may be found in RG40. The actual code will not be shown here but the logic of the MFTST routine is described under <u>Processing Multiple Input Files</u>. The reader should be aware that Chart KA is a <u>logical</u> chart only and does not directly correspond to the sequence of instructions found in RG40.

The EOFTS Routine

The end of file test (EOFTS) routine is generated in place of MFTST if there are no secondary files specified in the source program. For reference, the following is the actual code of this routine.

```
0A0C 0  C0CB       0722    EOFTS LD      STATU     EOF REQUESTED ON PRIMARY    Y4007220
0A0D 00 4C2800B6   0723    INT20 BN      PRIEO-MFTST  YES                      Y4007230
0A0F 0  1004       0724    INT53 SLA     4         WAS EOF SENSED ON SECONDARY Y4007240
0A10 00 4C1000CA   0725    INT21 BNN     PROCE-MFTST  NO, GO PROCESS           Y4007250
0A12 00 C400014F   0726    INTR5 LD   L  MRDIS     IS INTERNAL MR ON           Y4007260
0A14 00 4C1000F4   0727    INTR6 BNN     OBEOJ     NO GO TO EOJ IN OVER HEAD   Y4007270
0A16 0  701A       0728    INTR7 B       PROCE     GO  PROCESS                 Y4007280
0A17 0  C0C0       0729    SECTS LD      STATU     HAS                         Y4007290
0A18 0  E0C1       0730          AND     X1800     EOF                         Y4007300
0A19 0  90C0       0731          S       X1800     OCCURRED  ON BOTH           Y4007310
0A1A 00 4C1800AB   0732    INTR8 BZ      INTR5-MFTST  YES                      Y4007320
0A1C 0  7014       0733    INTR9 B       PROCE     GO  PROCESS                 Y4007330
0A1D 0  C0BA       0734    PRIEO LD      STATU     EOF  REQUESTED              Y4007340
0A1E 0  1001       0735          SLA     1         ON SECONDARY                Y4007350
0A1F 00 4C2800B0   0736    INT22 BN      SECTS-MFTST  YES,  CHECK BOTH         Y4007360
0A21 0  1002       0737    INT23 SLA     2         WAS EOF SENSED ON PRIMARY   Y4007370
0A22 00 4C1000CA   0738    INT24 BNN     PROCE-MFTST  NO,  GO PROCESS          Y4007380
0A24 00 C400014F   0739          LD   L  MRDIS     MATCHING                    Y4007390
0A26 0  1004       0740          SLA     4         RECORDS  USED               Y4007400
0A27 00 4C2800F4   0741          BN      OBEOJ     NO, GO TO EOJ IN OVERHEAD   Y4007410
0A29 00 C400014F   0742          LD   L  MRDIS     IS THERE A  MATCH           Y4007420
0A2B 00 4C1000F4   0743          BNN     OBEOJ     NO, GO   TO  EOJ            Y4007430
0A2D 0  C0AA       0744    INT48 LD      STATU     WAS EOF  SENSED             Y4007440
0A2E 0  1004       0745          SLA     4         ON  SECONDARY               Y4007450
0A2F 00 4C2800F4   0746          BN      OBEOJ     YES,  END  OF  JOB          Y4007460
0A31 0  C100       0747    PROCE LD     1 0        GET ADDR OF GET ROUTINE     Y4007470
0A32 00 D4000001   0748          STO  L  1         PUT IN XR1                  Y4007480
0A34 0  C106       0749          LD     1 DSDTF    GET  DTF  ADDRESS           Y4007490
0A35 00 D4000001   0750          STO  L  1         PUT IN XR1                  Y4007500
0A37 0  C103       0751          LD     1 3        GET IO AREA ADDRESS         Y4007510
0A38 00 D4000003   0752          STO  L  R2        PUT IN SIM REG              Y4007520
0A3A 00 C4000023   0753    LDLF  LD   L  CTLFD                                 Y4007530
0A3C 00 D4000010   0754          STO  L  R15       PUT IN SIM REG              Y4007540
0A3E 00 C4000000   0755    RI1   LD   L  ADDRI     GET RI ADDR FROM LOWFLD     Y4007550
0A40 0  E09F       0756          AND     Z3FFF     ZERO EOF BITS               Y4007560
0A41 00 D4000000   0757    RI2   STO  L  ADDRI     REPLACE IT                  Y4007570
0A43 00 4C000093   0758          B    L  PRORC     GO TO OVERHEAD              Y4007580
                   0759    **                                                  Y4007590
```

Figure 16. Object Code of the EOFTS Routine

When this routine receives control, it tests the high-order bit of the record identifying indicator address contained in Low Field (Word4). If the bit is on, EOF has occurred and a branch is taken to the instruction labeled EOJBK in the Fixed Driver. If the bit is off, the address of the control level compare routine (Word2) is taken from the Function Address Table and placed in PR15. A branch is then taken to PRORC in the Fixed Driver.

The MFEXT Routines

A matching fields extraction (MFEXT) routine is generated for each record type specified on the input specifications of the source program. Each routine extracts the matching fields for its record type and places them in the hold area of the proper processing block (PPB, S1PB, etc). The hold area is then compared to the hold area in the Low Field Block. If a collating sequence error occurred, a branch is taken to ERR+10 in the Fixed Driver.

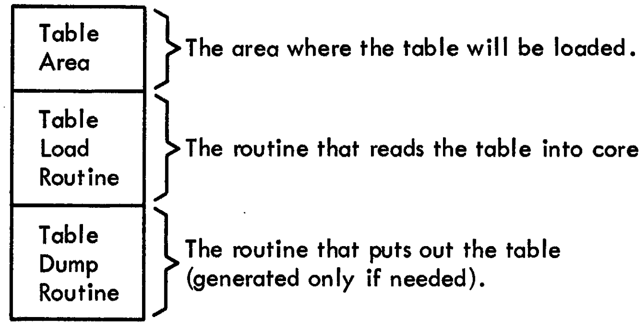
After extracting the fields, the routine branches into the MFTST routine. If the record type has no M1-M9 fields, the routine consists solely of a branch to the MFTST routine.

If there are no secondary files present, but M1-M9 fields are specified for the primary file, the MFEXT routine(s) places the fields in the PS hold area, and branches to the EOFTS routine.

CORE STORAGE ALLOCATION

Now that tables, work areas, and some of the main routines of RPG object programs have been discussed, it should be helpful to see where all these items appear in core storage when an object program is in residence. Some of the routines shown on the Core Storage Allocation Map (Figure 17) have not as yet been discussed, and, before tracing an object program through a full cycle, each section of the core map will be reviewed briefly. The numbers in the narrative correspond with the numbers in Figure 17.

1. Pseudo registers PR0-PR15: These 16 words are present in every RPG object program and are used for passing addresses and other parameters between the various object program routines.
2. CNTRL B L START: This long branch instruction is the entry point into the object program. START is a label in the OPEN/CLOSE routine.
3. Function Address Table (FAT): This table is present in every RPG object program and contains the address of various RPG routines (see Table 11).
4. Fixed Driver: This routine is generated in every RPG object program and functions as the main linkage driver for the entire object program.
5. Assigned indicators: Indicators MR, OO, OF, OV, 1P, L0-L9, LR, and H1-H9 are always generated in this area and in this order. These indicators are followed by any other indicators defined in the source program. Each indicator uses one word of core storage. If on, the indicator will be set to 0001. If off, the indicator setting will be 0000.
6. Assigned fields: Space is allocated here for any fields defined in the source program. One word is allocated for each character position in the field; in addition, an attribute word is generated for each field. (Refer to Appendix A: Object Time Data Format for a complete description of object time data fields.)
7. Assigned literals: Space is allocated here for any literals defined in the source program.
8. Control level hold areas (Old Hold and New Hold): If control levels were specified in the source program, the hold areas for control level processing are generated here.
9. Card and printer IODs: One input/output driver (IOD) is generated for every input or output file specified in the source program. The IOD for a given file is the routine which provides linkage to a library subroutine which will perform the actual I/O operation.
10. Disk IODs: If disk files are specified in the source program, their IODs are generated immediately following any card and/or printer IODs.
11. Tables, table load and dump routines: For every table file specified in the source program the following storage areas will be allocated.



12. Record Address File (RAF) routine: If the primary or one of the secondary files is to be processed by means of a record address file, the RAF routine will be generated here. Further information on RAF processing is contained in Processing With an RA File.
13. Chaining routines: If any files specified in the source program are to be retrieved by C1, C2, or C3 chaining, the chaining routine for each of these (a routine for C1, if used; another routine for C2, if used; etc.) will be generated here. A description of chaining routine logic may be found in the section Processing a File by C1, C2, or C3 Type Chaining. If no C1, C2, or C3 type chaining is specified, a branch to detail calculations is generated.
14. Move input fields (INPF) routines: A move input fields routine is generated for every input record type specified in the source program. Each routine extracts the defined fields from the input record and moves them to the field areas allocated.
15. Control level compare (COMP) routine: If control levels are specified in the source program, the routine that determines whether a level break has occurred and then sets the appropriate indicators is generated immediately following the INPF routines.
16. Numeric Sequencing routine: Space is allocated here for the numeric sequencing routine if input record types have numeric sequencing specified in the source program. A separate routine is generated for each file which has this specification. This routine ensures that the proper record type has been read at the proper time. More information is contained in the "NUMERIC SEQUENCING" section of this publication.

Generated by:		
	0.	One word unused
RG58	1.	Pseudo registers PR0-PR15
RG58	2.	CNTRL B L START
RG58	3.	Function Address Table (FAT)
RG58	4.	Fixed Driver
RG10	5.	Assigned indicators
RG12	6.	Assigned Fields
RG14	7.	Assigned Literals
RG17	8.	Old Hold and New Hold for Control Levels
RG24	9.	Card and printer IODs
RG28 ISAM RG26 Sequential	10.	Disk IODs
RG32	11.	Tables, table load and dump routines
RG34	12.	RAF routine
RG34	13.	Chaining routines
RG36	14.	Move input fields (INPF) routines
RG38	15.	Control level compare routine
RG38	16.	Numeric Sequencing routine
RG38	17.	Determine record type (INPR) and control level extraction routines
RG40	18.	Low Field and processing blocks
RG40	19.	EOFTS or MFTST routine
RG40	20.	MFEXT routines
RG42	21.	File Input Tables
RG42	22.	GET routines
RG44	23.	LOKUP and CHAIN routines
RG46	24.	Detail, then Total calculations
RG52	25.	Move output fields routines
RG54	26.	Central Output Driver (COD)
RG54	27.	Test indicators routines
RG54	28.	Output Tables (DTAB, TOTAB, OTAB and EXTAB)
RG58	29. Open/ Close Routines	OPEN branch table Linkage to load table routines Linkage to heading and detail lines CLOSE branch table Linkage to EOJ
	30.	Library subroutines
	31.	Core Storage left over
	32.	Transfer Vector

Figure 17. Core Storage Allocation MAP

17. INPR routines: An INPR (determine record type) routine is generated for each record type specified in the source program. These routines check the record ID characters in the associated input record to determine if the record type is being processed. If so, the address of the resulting indicator is passed back to the GET routine.
    If a record type has control level fields, the control level extraction routine for that record type is generated immediately following the INPR routine for that same record type. The control level extraction routine takes the level fields from the input record, places them into the New Hold area, and branches to the control level compare routine.
18. Low Field and processing blocks: Space is allocated here for the generation of Low Field and the file processing blocks (see Tables and Work Areas for further information).
19. EOFTS or MFTST routine: If no secondary files are present, Low Field is followed by the EOFTS routine. If secondary files are present, the processing blocks allocated for them are followed by the matching fields test (MFTST) routine which carries its own EOF code.
20. MFEXT routines: A matching fields extraction routine is generated for each input record type. If the record type has no matching fields specified, a branch instruction is generated to the MFTST routine.
21. File Input Tables: An FIT is generated for each input file specified in the source program (refer to Tables and Work Areas for further information).
22. GET routines: A GET routine is generated here for each primary and secondary file specified in the source program.
23. LOKUP and CHAIN routines: If LOKUP is specified in the source program, the LOKUP routine is generated here. If CHAIN is specified, its corresponding routine would also be generated here.
24. Detail calculations: All detail calculations specified in the source program are generated here.
    Total calculations: All total calculations specified in the source program are generated immediately following the detail calculations.
25. Move output fields routines: A move output fields routine is generated for each output line specified in the source program. Each routine moves the fields for its associated line into the output buffer.
26. Central Output Driver: The COD is the routine that receives control each time a line is to be put out. Further information is contained in the section Output Lines Routines.
27. Test indicators routines: A test indicators routine is generated for each output line specified in the source program. Each of these routines tests the indicators conditioning its associated output line and, if the indicators are on, provides linkage to perform services such as stacker select, or carriage control functions, if necessary, directly to the COD to put out the line.

    If the indicators are off, the Fixed Driver is given control.
28. Output Tables: Space is allocated here for the generation of the output tables, DTAB, TOTAB, OTAB, and EXTAB (see Tables and Work Areas).
29. OPEN/CLOSE Routine: This routine receives control from the CNTRL instruction. If disk files are present in the program, the routine links to the OPEN entry point of each disk IOD. After all files are opened, this routine links to the table load routines, if present, to load the tables. After the tables are loaded, a branch is made to the heading and detail lines entry in the COD. The OPEN/CLOSE routine also contains code which links to the CLOSE entry points of each disk IOD which culminate in EOJ.
30. Library Subroutines: All library subroutines used by the mainline program follow the OPEN/CLOSE routine. These routines are all described in the section Library Subroutines.

31.&32. Any unused core storage will fall between the library subroutines and the Transfer Vector (high core) used for linking to the subroutines.

TRACING AN OBJECT PROGRAM

In this section the operation of an object program will be traced through its full cycle. The source code that generated this object program is shown in Figure 18.

In following the step by step object program description, it may be helpful to refer to the preceding sections of this publication whenever clarification or additional information is necessary.

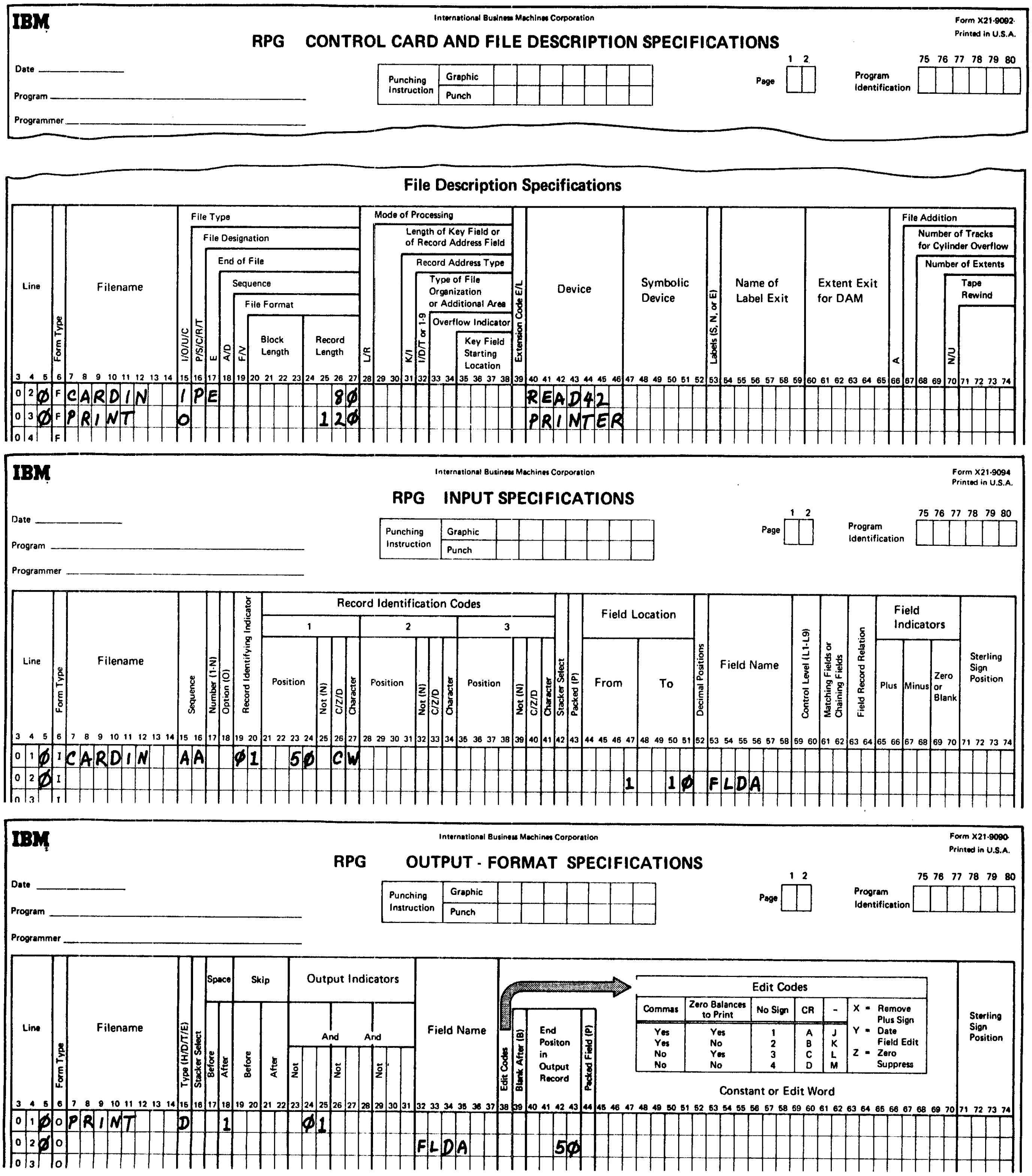


Figure 18. Typical Source Code for Object Program Generation

Initialization

- The monitor passes control to the instruction labeled CNTRL.
- CNTRL branches to START, which is the beginning of the OPEN/CLOSE routine. Since there are no disk files and no table files, this routine immediately links to the heading and detail lines entry point of the COD. This entry point is B0000.

Heading and Detail Lines

(STEP3)

- At B0000, PR8 is loaded with the address of B0001. PR10 is loaded with the address of ALPSE in the Fixed Driver. PR11 is loaded with the address of DTAB-3. A branch is then taken to ALPSE in the Fixed Driver.
- At ALPSE, PR11 is incremented by 3 to point at the first entry of DTAB. The address of the test indicators routine for the output line specified in the source program is taken from Word1 of the table entry and a branch is made to the routine.
- The output line in this program is conditioned by indicator 01, so the test indicators routine determines whether 01 is on. Since, at this time, an input record has not yet been read, 01 will be off. The test indicators routine will then return to ALPSE in the Fixed Driver.
- At ALPSE, PR11 is again incremented by 3, and now points at the dummy entry in DTAB since only one detail line is specified in the source program. Word1 of this dummy entry (address of B0020 in the COD) is taken and a branch is made to that address.
- B0020 is the entry point in the COD that is used for wrapping up heading and detail lines. The 1P indicator is turned off and a branch is taken to GETRC in the Fixed Driver.

Get Input Record

- At GETRC the LR and halt indicators are tested. None are on as yet, so all level indicators are set off and record identifying indicator 01 is set off. The address of the GET routine for the input file is then taken from Word1 of Low Field (LOWFD), placed in PR8, and a branch is taken to the GET routine.
- The GET routine then loads PR8 with the address of a two-word area. Word1 of that area contains the address of the input/output driver (IOD) for the input file, and Word2 contains the address of the File Input Table -4 for this input file. A branch is then made to ADDGT in the Fixed Driver.

STEP10)

- At ADDGT the File Input Table -4 address is placed in PR11, the IOD address is placed in PR15, and the contents of PR8 are placed in PR14. The address of the I/O area for the input file is taken from the IOD (Word4) and placed in PR2. The read entry point of the IOD is taken from wordz of the IOD, the address of INPSE in the Fixed Driver is placed in PR10, and a branch to the read entry point of the IOD is executed. Note that PR8 and PR14 still contain the address of the two-word storage area (which contains the address of the IOD and the File Input Table -4 for this file).
- For this object program, the IOD links to the library subroutine CARDO and the first card is read into the I/O area pointed at by PR2. The IOD then checks to determine whether EOF occurred. If not, a branch is taken to the contents of PR14+2. If EOF did occur, a branch is taken to the contents of PR14+8. Assuming that EOF did not occur, the resulting branch takes us into the GET routine at DTFA+2 (RETUR).

Determine Record Type

- At RETUR the address of LADD is placed in PR8 and a branch is taken to INPSE in the Fixed Driver. Note that PR11 still points at the File Input Table -4.
- At INPSE, PR11 is incremented by 4 to point at the first entry of the File Input Table. The address of the input record routine (INPR) is taken from the table and a branch is made to that address.
- The INPR routine starts by checking the 50th character of the record now in the input I/O area. If it is not a W, the routine will return to the Fixed Driver at INPSE. Assuming that the record read in does have a W as the 50th character, INPR loads PR8 with the address of record identifying indicator 01. INPR then branches back to the GET routine at the instruction labeled LADD.

(STEP15)

- At LADD the address of LOWFD is placed in PR10. Word3 of the File Input Table entry pointed to by PR11 is then placed in Word2 of LOWFD, and Word4 of the FIT entry is placed in Word3 of LOWFD. The record identifying indicator address (still in PR8) is placed in Word4 of LOWFD. (Note that if multiple input files were being processed, the preceding information would not be put in LOWFD at this time; it would be placed in the processing block associated with the GET routine that has control.)

  The routine then ORs the EOFWD with the record identifying indicator address and places the result back in Word4 of LOWFD. EOFWD initially contains /0000 if EOF is required for this file; /4000 if not required. When EOF actually does occur, the IOD which processes the input file sets the high-order bit of EOFWD to 1.

  If the high-order bit of EOFWD is on, a branch to the EOFTS routine is taken. In this case, EOF did not occur and the bit is still on so a branch will be taken to MFLNK in the Fixed Driver.

  Note that PR11 still points to the File Input Table entry for the record type being processed.

- At MFLNK, Word2 of the File Input Table (address of the MFEXT routine for this record type) is taken and a branch is executed to that address.

- Since there are no M1-M9 fields in this record type, the MFEXT routine for this file consists of a branch to EOFTS.

Test for Control Level Break

- At EOFTS the high-order bit of Word4 of LOWFD is tested (note that this bit was set by step15 above). If the bit is on, EOF has occurred and a branch is taken to EOJBK in the Fixed Driver.

  In this case, EOF did not occur and the two high-order bits of Word4 in LOWFD are zeroed out. Then the address of the control level compare (COMP) routine is taken from CTLFD in the Function Address Table and placed in PR15. A branch is then taken to PRORC in the Fixed Driver. Note that PR10 still contains the address of LOWFD.

- At PRORC the address of the record identifying indicator (01 for this record) is taken from LOWFD and the indicator is set on (/0001). A check is then made to determine if there is a COMP routine in this program. There is none since no control levels were specified in the source program, so a branch is taken to TOTSW in the Fixed Driver.

Total Calculations

- At TOTSW a check is made to determine if this is the first cycle of the object program. This time it is, so instead of branching to total calculations, a branch is executed to GETIF in the Fixed Driver.

- At GETIF the LR indicator is tested. It is not on at this point so Word3 (address of move input fields routine filled by step15) is taken from LOWFD and placed in PR8. The I/O area address is taken from the reader IOD and placed in PR2. The internal matching records indicator (0 at this time) is stored in the MR indicator and a branch is then taken to LABEO in the Fixed Driver.

Move Input Fields

- At LABEO a branch is executed via PR8 to the move input fields routine for the record type now in the I/O area.

- The move input fields routine then extracts FLDA from the I/O area and puts it in its assigned field area. This routine then obtains the address of the chaining routine from CHAN1 in the Function Address Table and branches to that routine.

Chaining Routine

- Chaining was not specified in this source program so only a B I DCALC instruction was generated. This branch goes to the detail calculations routine.

Detail Calculations

- No calculations were specified in the source program so only a B I HDAD was generated. This branch goes to B0000 which is the heading and detail lines entry point to the COD.

  The object program has now completed a full cycle and is back at step3 above. Upon entering step3 this time, however, output indicator 01 is on, so the line will be put out.

PROCESSING WITH AN RA FILE

In the object program just described there was only one input file. Suppose that the file resided on disk and it was decided to retrieve it randomly by using a record address file. In this instance, the source program might appear as follows (Figure 19).

Again referring to the previous object program, processing with an RA file would cause these changes:

1. The RAF routine would be included in the object program.
2. A card IOD would be generated for RAFIN followed by a disk IOD for DISKIN.
3. The constant labeled DTFA in the GET routine for the primary file would no longer point to the IOD for that file. Whenever a file is being processed via an RA file, this constant points at RAFAD-1 in the Function Address Table.

The program logic for this object program would be very similar to the logic of the previous program. Note, however, that at step10 a branch is not taken to the read entry point of the primary file IOD. Since DTFA now points to RAFAD-1, a branch is taken to the RAF routine.

The RAF routine will get a record address field and then link to the read entry point of the primary file IOD. This means, in effect, that any request for the primary file would be intercepted by the RAF routine.

The flowchart provided (Chart HA) illustrates the logic used by the RAF routine to process a sequential file randomly. The RAF routine is a pre-coded routine found in RG34.

File Description Specifications

Line	Form Type	Filename	File Type (I/O/U/C)	File Designation (P/S/C/R/T)	End of File (E)	Sequence (A/D)	File Format (F/V)	Block Length	Record Length	Mode of Processing (L/R)	Length of Key Field or of Record Address Field	Extension Code E/L	Device
02ø	F	RAFIN	I	R			F		8ø		4	E	READ42
03ø	F	DISKIN	I	P			F		8ø	R			DISK
04ø	F	PRINT	O				F		12ø				PRINTER

IBM International Business Machines Corporation Form X21-9091 Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch   Page 1 2   Program Identification 75 76 77 78 79 80

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table Name	Number of Table Entries Per Record	Number of Entries Per Table	Length of Table Entry	Packed (P)	Decimal Positions	Table Sequence (A/D)	Table Name (Alternating Table)	Length of Table Entry	Packed (P)	Decimal Positions	Table Sequence (A/D)	Comments
01ø	E			RAFIN	DISKIN													

IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch   Page 1 2   Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Record Identification Codes 1: Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Decimal Positions	Field Name
01ø	I	DISKIN	AA			ø1	5ø		C	W				
02ø	I										1	1ø		FLDA

IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch   Page 1 2   Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	Packed Field (P)	Constant or Edit Word
01ø	O	PRINT	D			1			ø1						
02ø	O									FLDA			5ø		

Edit Codes: Commas / Zero Balances to Print / No Sign / CR / -
Yes Yes 1 A J; Yes No 2 B K; No Yes 3 C L; No No 4 D M
X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress

Figure 19. Processing With an RA File

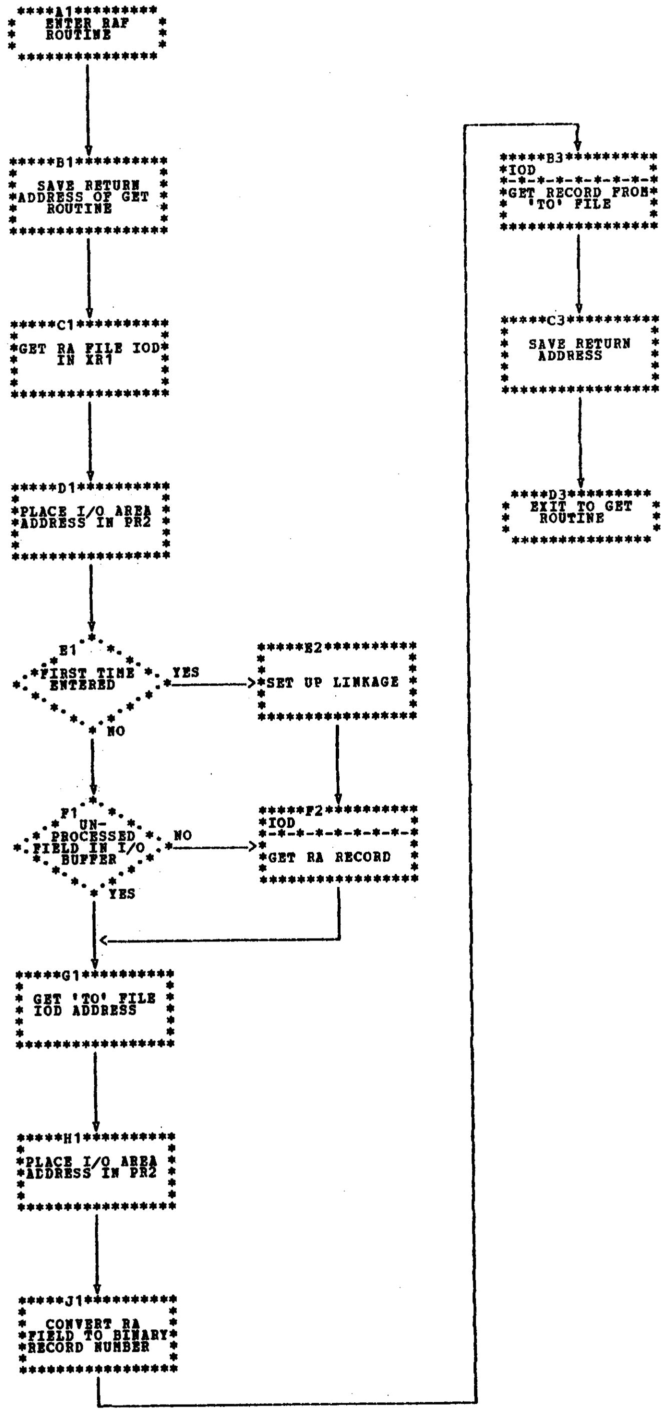


Chart HA. RAF Routine

PROCESSING BY C1, C2, OR C3 TYPE CHAINING

It has been stated before that a move input fields routine is generated for every input record type. These routines extract defined input fields from the I/O area and move them to their assigned areas. When a move input fields routine has completed its function, a branch is always taken to the chaining routine address, which is CHAN1 in the Function Address Table, whether chaining has been specified or not. If one of the extracted fields is a chaining field, the move input fields routine sets on its corresponding internal indicator (C1, C2, or C3).

Then, when the chaining routine receives control, it accomplishes the following:

- gets the chained record
- determines the record type
- moves the chained record's input fields.

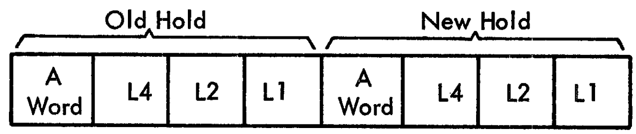
The move input fields routine will then return to the beginning of the chaining routine and the process will be repeated if any of the indicators C1, C2, or C3 are on. If none of the three indicators are on, the chaining routine links to detail calculations.

Refer to Figure IA for the logic of the chaining routine.

CONTROL LEVEL PROCESSING

A short control level extraction (CLEV) routine is generated for any record type that is specified with L1-L9 fields. A CLEV routine simply extracts the level fields from the I/O area and places them in their proper location in New Hold.

If, for example, an RPG source program had control levels L1, L2, and L4 specified, Old Hold and New Hold would appear as follows:



Once a control level extraction routine is finished placing the level fields, it branches to the control level compare (COMP) routine. Chart JA shows how COMP would be generated for a program which had control levels L1, L2, and L4 specified.

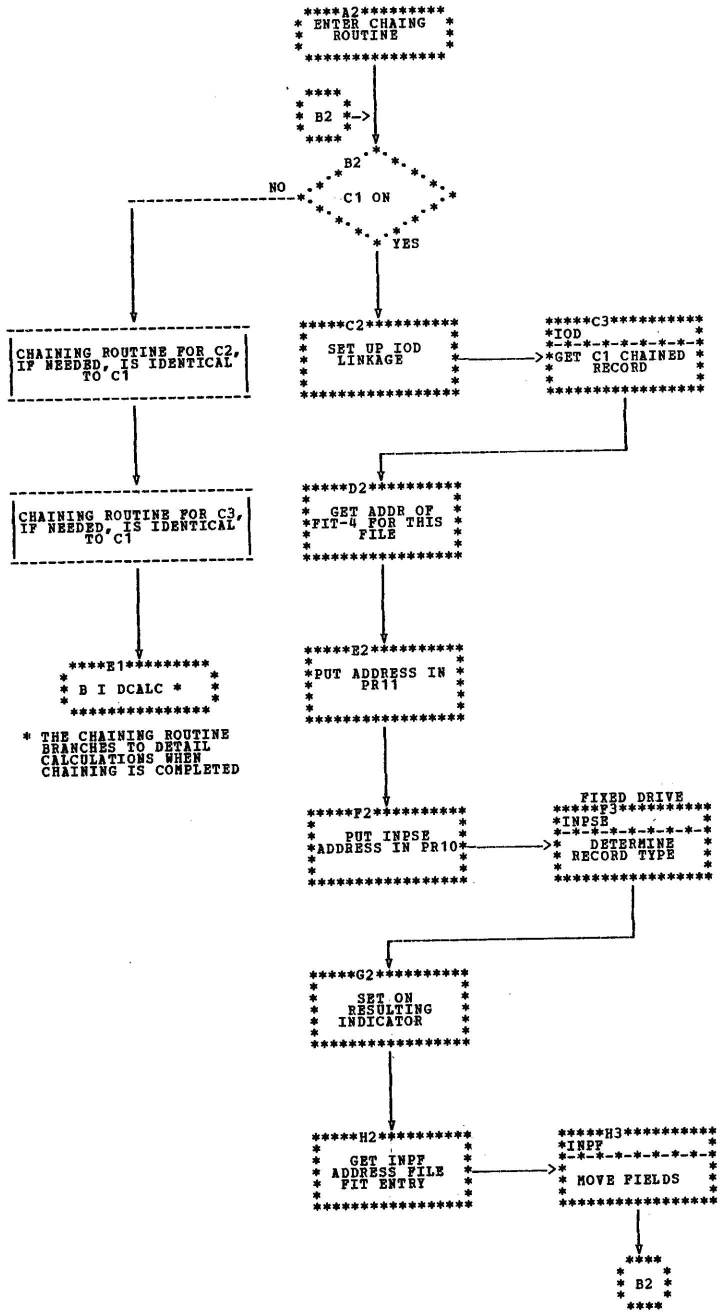


Chart IA. Logic of the Chaining Routine

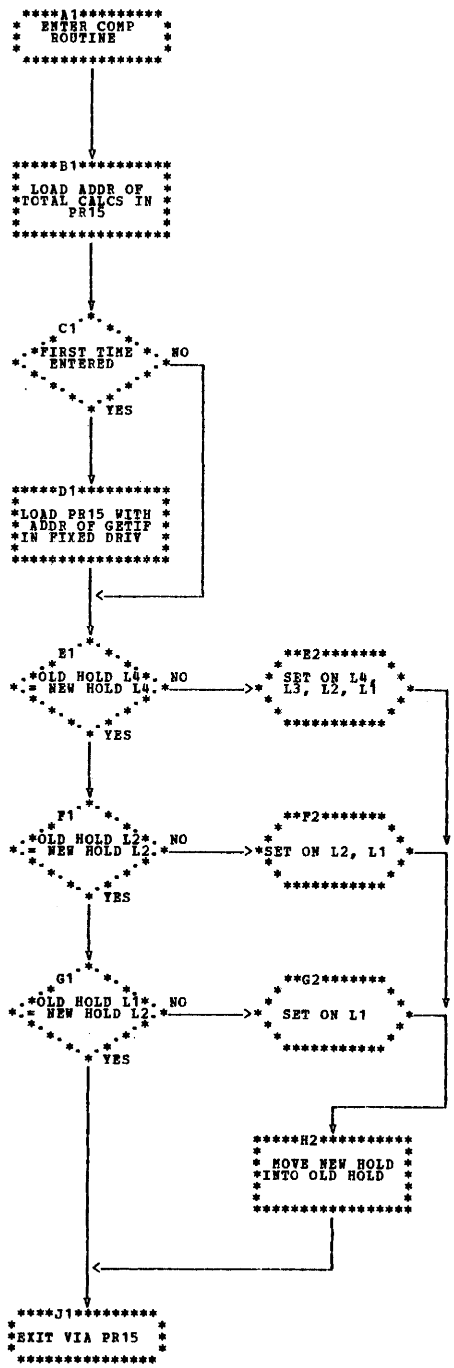


Chart JA COMP Routine

IBM — International Business Machines Corporation — Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Field Name	Matching Fields or Chaining Fields
01Ø	I	PRIM	AA			Ø1	1		C	P				
02Ø	I										1	1Ø	FLD1	M1
03Ø	I	SEC	BB			Ø2	1		C	S				
04Ø	I										11	2Ø	FLD2	M2
05	I													

Figure 20. Processing Multiple Input Files

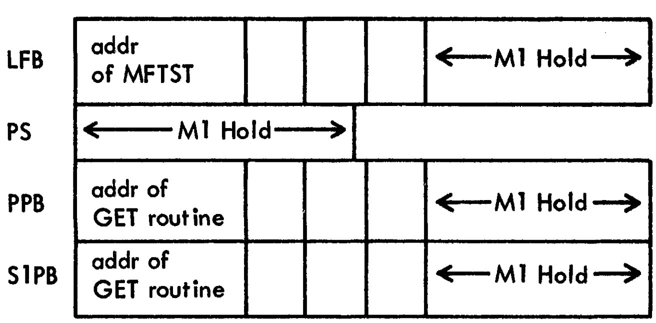
PROCESSING MULTIPLE INPUT FILES

Previous sections of this publication have described how the object program performs output operations, input operations for a single file, determines record type, and handles control level, chained and RA file processing. At this time it should be helpful to describe how the object program processes multiple input files with matching records.

The input specifications in Figure 20 describe a source program that contains a primary file and one secondary file.

Figure 21 shows the routines that would be generated to process these files.

A. The Low Field Block, PS, primary processing block (PPB), and one secondary processing block (S1PB) are generated. They would appear in core storage as follows:



B. The MFTST routine would be generated immediately following S1PB.

C. A matching field extraction (MFEXT) routine for record type 01 and another for record type 02 would be generated after the MFTST routine.

D. Two File Input Tables, one for each input file, are generated and follow the MFEXT routines.

E. Two GET routines are generated. Word1 of PPB points to the GET routine for the primary file; Word1 of S1PB points to the GET routine for the secondary file.

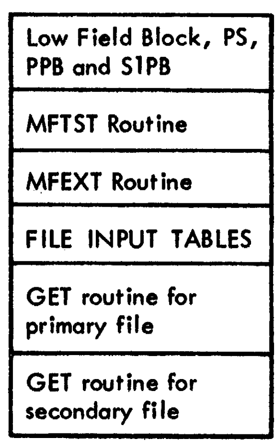


Figure 21. Routines Generated to Process Multiple Input Files

Steps 1 through 13 shall attempt to describe the logic involved in getting an input record and selecting it for processing. In following this description, it again may be helpful to refer to preceding sections of this publication whenever clarification or additional information is necessary.

1. The instruction labeled GETRC in the Fixed Driver receives control from the COD after heading and detail lines are completed.
2. Word1 of the Low Field Block is obtained and placed in PR8. (Initially, this word contains the address of MFTST.)
3. A branch is made via PR8 to the MFTST routine (see Chart KA for a better understanding of the MFTST routine).
4. At MFTST, since this is the first time through the routine, the entire PPB is moved into the Low Field Block and a branch is made back to GETRC in the Fixed Driver.
5. At GETRC, Word1 of Low Field (which now contains the address of the primary file GET routine) is taken and placed in PR8, and a branch to the GET routine is then executed.
6. PR8 is then loaded with the address of DTFA (refer to The GET Routines) and a branch is made to ADDGT in the Fixed Driver.
7. At ADDGT the address of the File Input Table -4 is placed in PR11. Linkage to the IOD is initialized and the address of INPSE is placed in PR10. A branch is then taken to the read entry point of the IOD for the primary file.
8. The GET routine is again entered, this time at RETUR, which loads PR8 with the address of LADD and then branches via PR10 to INPSE in the Fixed Driver.
9. At INPSE, PR11 is incremented by 4 to point at the first entry of the primary File Input Table. The address of the INPR routine is obtained and a branch is made to the INPR routine which will check column 1 of the input record for a P. If a P is present (for this example, assume that it is), PR8 is loaded with the address of record identifying indicator 01 and a branch is made back to the GET routine at the instruction labeled LADD.
10. At LADD the address of Low Field is placed in PR10 and Word3 of the File Input Table entry is placed in Word2 of the PPB. Word4 of the FIT entry is then placed in Word3 of the PPB. The address of record identifying indicator 01 is placed in Word4 of the PPB. Then, since EOF has not occurred, a branch is taken to MFNK in the Fixed Driver.
11. At MFLNK the address of the MFEXT routine for the record type identified with indicator 01 is taken from Word2 of the File Input Table entry still pointed to by PR11, and a branch is made to that extraction routine.
12. The MFEXT routine then takes F1D1 from the I/O area and puts it in the PPB hold area. Next, MFEXT compares the PPB hold area with the Low Field hold area in order to check for collating sequence errors. In this case there are none, so a branch is taken to the MFTST routine.
13. When the decision "Has a record been read from all the files?" has been made, the decision will be "no", therefore S1PB is moved into the Low Field Block and a branch is made to GETRC in the Fixed Driver.

The program is now back at Step5 of the cycle, but now the address of the GET routine for the secondary file is in Word1 of Low Field.

At this point a primary record has been read into core storage, its record type has been determined, and the addresses of the control level extraction routine, the move input fields routine, and the resulting indicator have been placed in the PPB. Also, the M1 field has been extracted and placed in the PPB hold area.

Now, steps 5 through 12 will be executed again in order to read a record from the secondary file, determine its record type, and fill S1PB with the proper information. This leads back to the MFTST routine that will decide which record should be processed. The information contained in the processing block for the chosen record will be moved to the Low Field Block, and a branch will be taken to PRORC in the Fixed Driver where normal processing will continue as it would for a single input file.

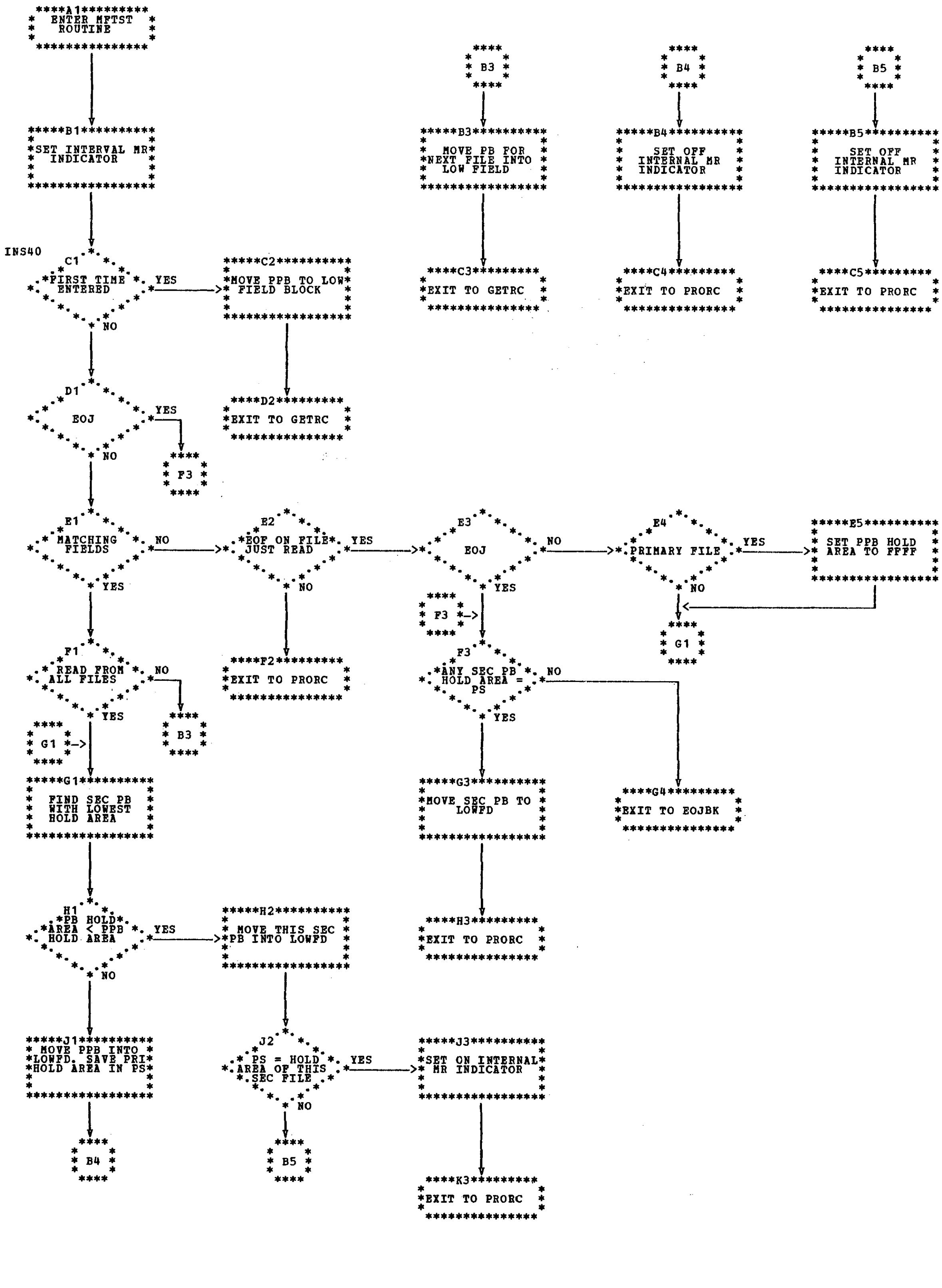


Chart KA. MFTST Routine

NUMERIC SEQUENCING

If a numeric record type is present, NUSEQ routine supplies object code for a sequence-check of the various record-types (columns 15-16 of the input specification sheet). If all records have an alpha entry in columns 15-16 of the input specification, this routine is not generated.

The address of routine NUSEQ is placed in Q (sequence) position of the compressed record (Table 8). The following object code is put out for NUSEQ.

			1003	*				
0A8C	00	C4000000	1004	NUSEQ	LD	L	R11	
0A8E	00	D400000A	1005		STO	L	R9	
0A9C	00	6C000010	1006		STX	L	R15,0	BALR 15, 0
0A92	00	C4000C00	1007	NOPX	LD	L	*-*	USED AS A SWITCH
0A94	00	6580000A	1008		LDX	11	R9	
0A96	0	0101	1009		LD	1	1	GET 2ND ENTRY
0A97	00	D400000C	1010		STO	L	R11	PUT IN R11
0A99	0	7000	1011	SW	B		*	BRANCH-NCP
0A9A	00	668000CC	1012	PASS	LDX	I2	R11	
0A9C	0	C201	1013		LD	2	1	IS IT
0A9D	0	9205	1014		S	2	5	AN OR,QQ
0A9E	0	7204	1015		MDX	2	4	BUMP POINTER
0A9F	00	6E00000C	1016		STX	L2	R11	
0AA1	00	4C180A9A	1017	A	BZ		PASS	ITS AN OR
0AA3	0	C014	1018	SWOFF	LD		BC0	SET
0AA4	0	D0F4	1019		STO		SW	TO NCP
0AA5	00	C400000C	1020	P4	LD	L	R11	
0AA7	0	D101	1021		STO	1	1	
0AA8	00	66000120	1022		LDX	L2	ERR1	
0AAA	00	6E00000F	1023		STX	L2	R14	
0AAC	00	6680000C	1024		LDX	I2	R11	
0AAE	0	C201	1025		LD	2	1	
0AAF	0	900A	1026		S		X5C5C	END OF FILTABS
0AB0	00	4C200033	1027		BNZ		ALPSE+2	NO, GO TO OVERHEAD
0AB2	00	6680000A	1028		LDX	I2	R9	
0AB4	0	C202	1029		LD	2	2	
0AB5	00	D400000C	1030		STO	L	R11	
0AB7	0	70ED	1031		B		P4	GO BACK
0AB8	0	7000	1032	BC0	B		*	OBJECT TIME SWITCH
0AB9	0	7009	1033	BCF	B		*+9	OBJECT TIME SWITCH
0ABA	0	5C5C	1034	X5C5C	DC		/5C5C	CHECK END OF FILTABS
0396	0		1035	HOLD9	EQU		/38E+8	
015E	0		1036	L9AD	EQU		INTMR+15	
			1037	*				

Figure 22. Object Code Put Out for Numeric Sequencing (Part 1 of 3)

If a numeric code is assigned under Sequence (columns 15-16), an entry of 1 or N must be made in Number (column 17), and the following object code is put out.

1 Mandatory
(Column 17 is 1, and column 18 is blank)

```
followed by 'OR'        CLI     LD      L       *-*         IS FRR IND. ON
                        *                                          BZ      *-*
                                BZ              *-*         NO, DONT EXTRACT

                        RICR            (record identifying code routine)

                        BALR    LDX     I2      R8          GET BRANCH ADDRESS
                                LDX     L1      RI          GET ADDR OF INDR ADDR
                                STX     L1      R8          SAVE IN PR8
                                B       L2      0           RETURN
                        RI      DC              0           RESULTING INDR ADDR

not followed by 'OR'    CLI     LD      L       *-*         IS FRR IND. ON
                        *                                          BZ      *-*
                                BZ              *-*         NO, DONT EXTRACT
                        RICR
                        BALR    LDX     I2      R8          GET BRANCH ADDRESS
                                LDX     L1      RI          GET ADDR OF INDR ADDR
                                STX     L1      R8          SAVE IN PR8
                                B       L2      0           RETURN
                        RI      DC              0           RESULTING INDR ADDR
```

1 Optional
(Column 17   1, and column 18   0)

```
followed by 'OR'        CLI     LD      L       *-*         IS FRR IND. ON
                        *                                          BZ      *-*
                                BZ              *-*         NO, DONT EXTRACT
                        RICR
                        BALR    LDX     I2      R8          GET BRANCH ADDRESS
                                LDX     L1      RI          GET ADDR OF INDR ADDR
                                STX     L1      R8          SAVE IN PR8
                                B       L2      0           RETURN
                        RI      DC              0           RESULTING INDR ADDR

not followed by 'OR'    CLC     DN              RGCMP       LIBF TO RGCMP
                                DC              *-*         OLD HOLD - FACTOR 1
                                DC              *-*         NEW HOLD - FACTOR 2
                                LD        3     123         GET RETURN CODE
                        BNE     BNZ             *-*         BRANCH IF NOT EQUAL TO

                        RICR
                        BALR    LDX     I2      R8          GET BRANCH ADDRESS
                                LDX     L1      RI          GET ADDR OF INDR ADDR
                                STX     L1      R8          SAVE IN PR8
                                B       L2      0           RETURN
                        RI      DC              0           RESULTING INDR ADDR
```

Figure 22. Object Code Put Out for Numeric Sequencing (Part 2 of 3)

N Mandatory
(Column 17   N, and column 18   blank)

```
followed by 'OR'         CLI      LD       L      *-*         IS FRR IND. ON
                         *                                           BZ      *-*
                                  BZ              *-*         NO, DONT EXTRACT

                         RICR
                         MVI      LDX      L1     /F0         SET INSTRUCTION IN
                                  STX      L1     *-*         NUSEQ TO FO
                         OI       LD       L      *-*         LD BCF IN NUSEQ
                                  STO      L      *-*         STO INTO SW IN NUSEQ
                         BALR     LDX      I2     R8          GET BRANCH ADDRESS
                                  LDX      L1     RI          GET ADDR OF INDR ADDR
                                  STX      L1     R8          SAVE IN PR8
                                  B        L2     0           RETURN
                         RI       DC              0           RESULTING INDR ADDR

not followed by 'OR'     CLI      LD       L      *-*         IS FRR IND. ON
                         *                                           BZ      *-*
                                  BZ              *-*         NO, DONT EXTRACT

                         RICR
                         OI       LD       L      *-*         LD BCF IN NUSEQ
                                  STO      L      *-*         STO INTO SW IN NUSEQ
                         BALR     LDX      I2     R8          GET BRANCH ADDRESS
                                  LDX      L1     RI          GET ADDR OF INDR ADDR
                                  STX      L1     R8          SAVE IN PR8
                                  B        L2     0           RETURN
                         RI       DC              0           RESULTING INDR ADDR
```

N Optional
(Column 17   N, and column 18   0)

```
followed by 'OR'         OI       LD       L      *-*         LD BCF IN NUSEQ
                                  STO      L      *-*         STO INTO SW IN NUSEQ

                         CLI      LD       L      *-*         IS FRR IND. ON
                         *                                           BZ      *-*
                                  BZ              *-*         NO, DONT EXTRACT

                         RICR
                         BALR     LDX      I2     R8          GET BRANCH ADDRESS
                                  LDX      L1     RI          GET ADDR OF INDR ADDR
                                  STX      L1     R8          SAVE IN PR8
                                  B        L2     0           RETURN
                         RI       DC              0           RESULTING INDR ADDR

not followed by 'OR'     NI       LD       L      *-*         CHANGE SW
                                  STO      L      *-*         TO  B *
                         CLI      LD       L      *-*         IS FRR IND. ON
                         *                                           BZ      *-*
                                  BZ              *-*         NO, DONT EXTRACT

                         RICR
                         OI       LD       L      *-*         LD BCF IN NUSEQ
                                  STO      L      *-*         STO INTO SW IN NUSEQ
                         BALR     LDX      I2     R8          GET BRANCH ADDRESS
                                  LDX      L1     RI          GET ADDR OF INDR ADDR
                                  STX      L1     R8          SAVE IN PR8
                                  B        L2     0           RETURN
                         RI       DC              0           RESULTING INDR ADDR
```

Figure 22. Object Code Put Out for Numeric Sequencing (Part 3 of 3)

OBJECT PROGRAM FLOWCHART

The following expanded flowchart (Chart LA) shows the logic of the RPG Object Program.

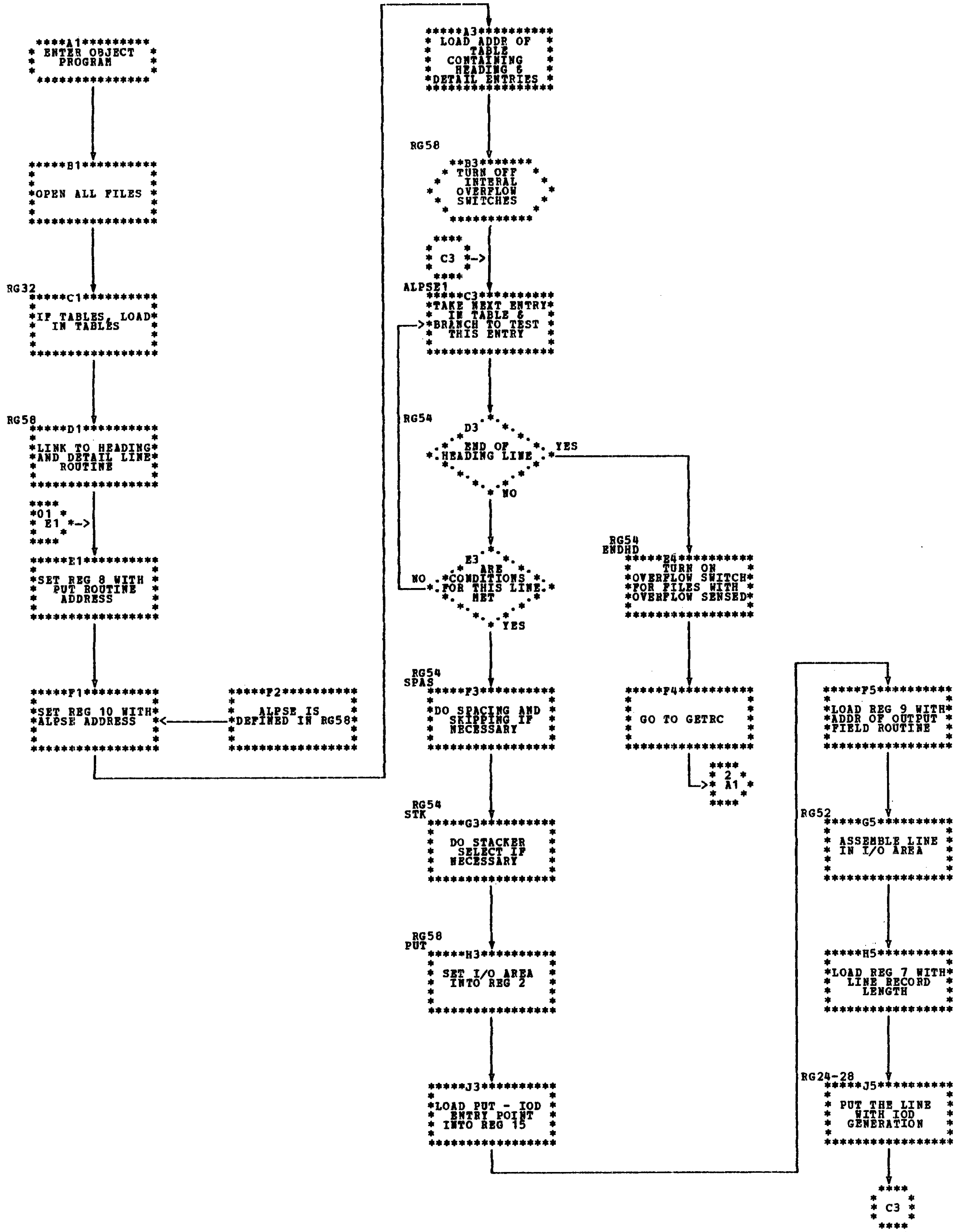


Chart LA. RPG Object Program (Part 1 of 4)

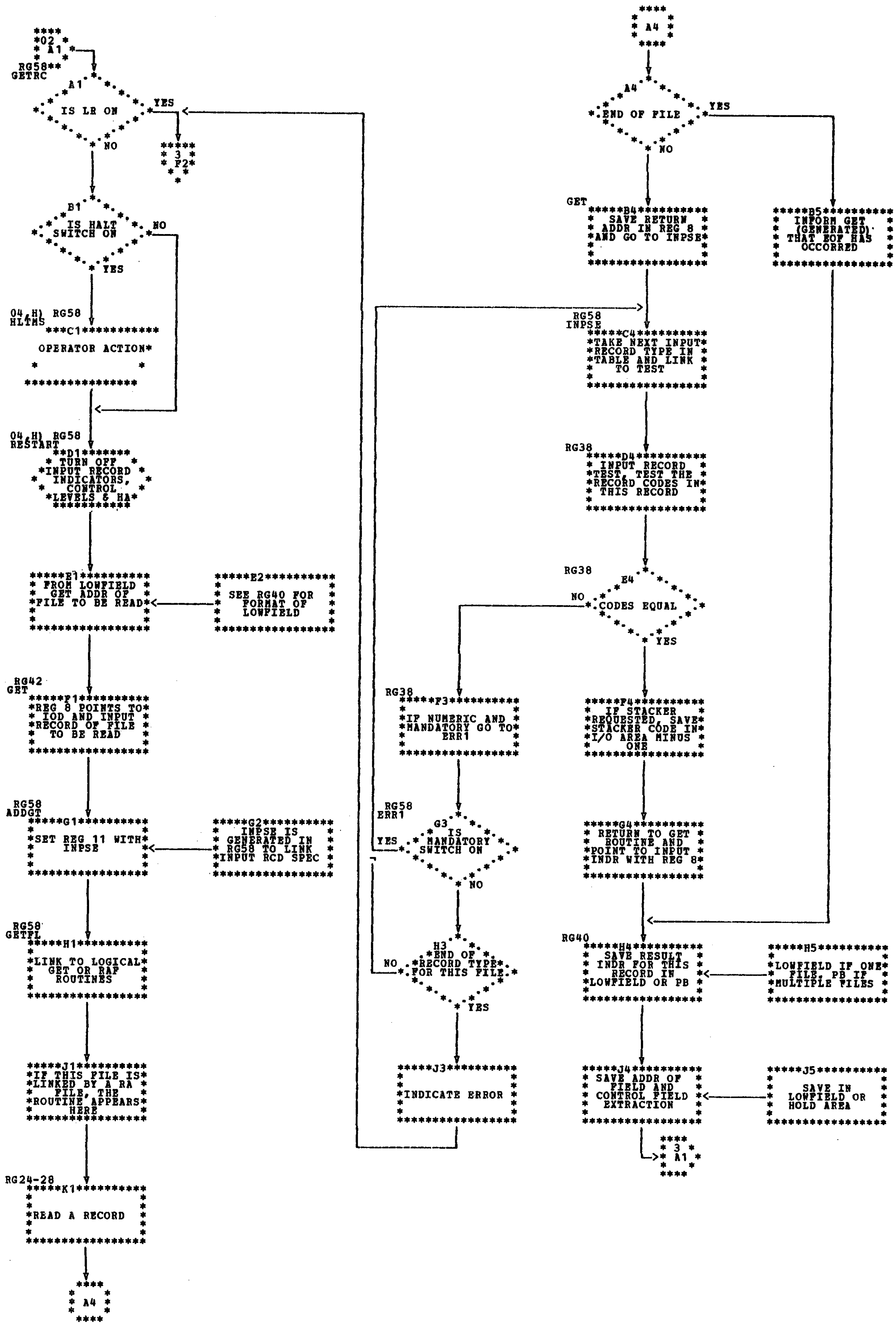


Chart LA. RPG Object Program (Part 2 of 4)

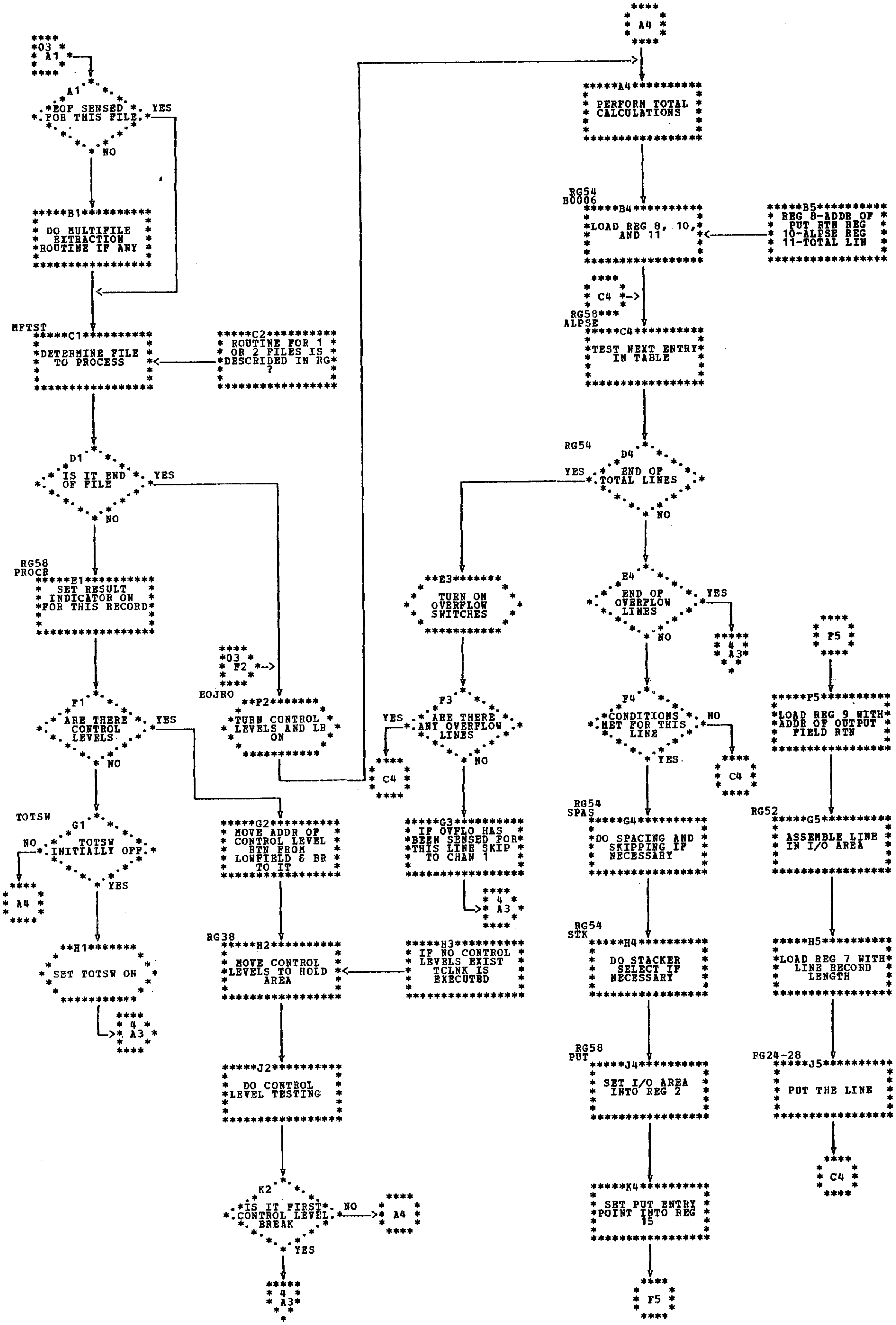


Chart LA. RPG Object Program (Part 3 of 4)

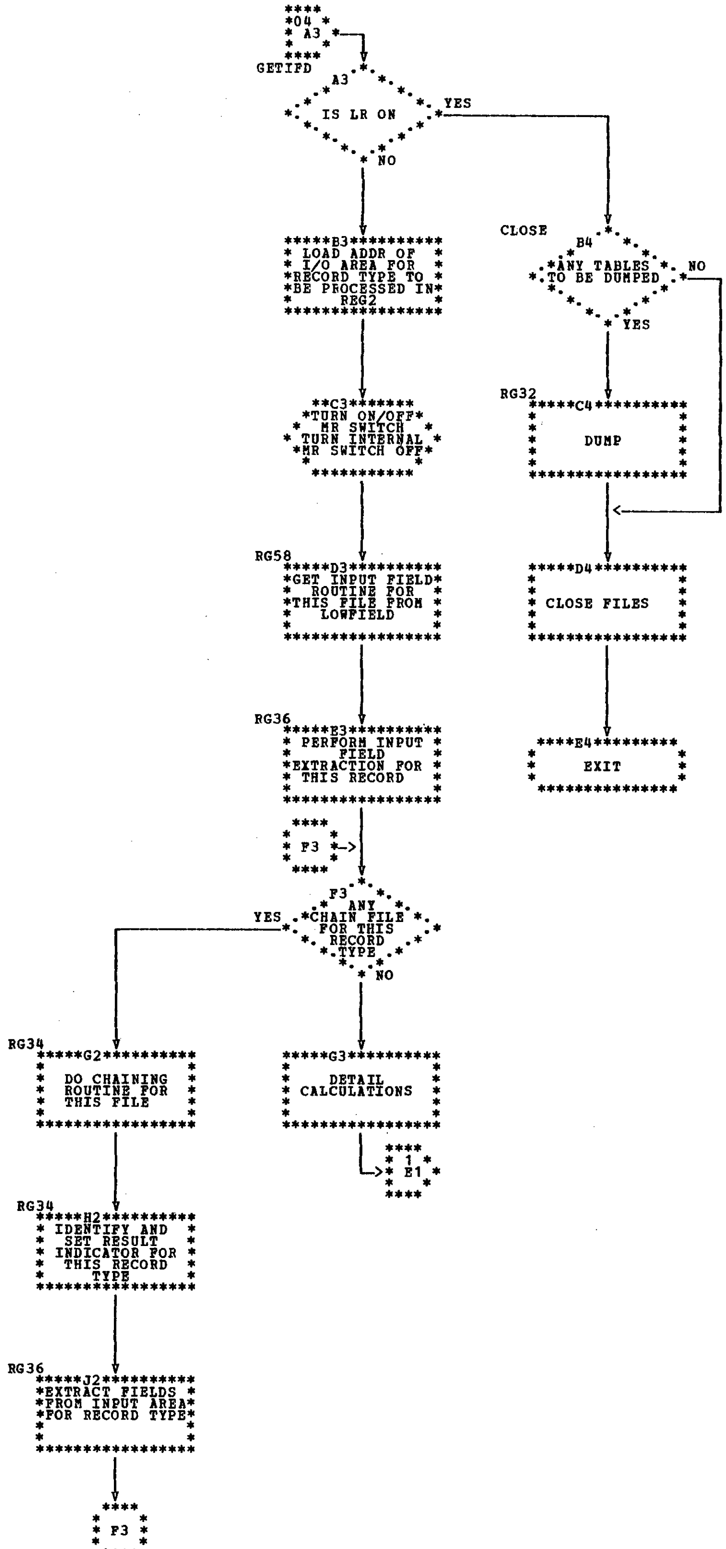


Chart LA. RPG Object Program (Part 4 of 4)

LIBRARY SUBROUTINES

This section contains flowcharts of the library subroutines which can be linked to by the object program generated by the 1130 RPG compiler. The library subroutines, which are assembled by the system in its library, are put in core storage by the core load builder, after it puts the mainline object code into storage, as shown in Figure 23.

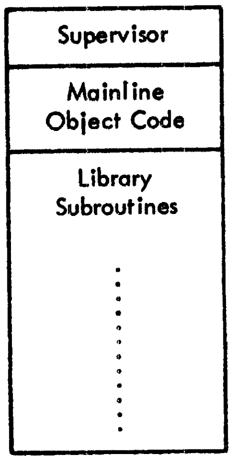


Figure 23. Location of the Library Subroutines in an Object-Time Core Load

The library subroutines are placed in storage in the order in which they are called by the object code routines. A subroutine will be stored only once, however, regardless of how many times it is to be called. Table 12 shows which subroutines are called by each object code routine.

Routines	Subroutines
FIXED DRIVER	RGERR
MOVE INPUT FIELDS	RGMV1 RGMV5 RGSI2 RGSI3 RGSI4 RGSI5 RGSTI
MOVE OUTPUT FIELDS	RGMV2 RGSTO RGEDT RGADD RGBLK
RAF routine	RGMV1 RGCVB ISET1
CHAINING routine	RGCVB
CONTROL LEVEL COMPARE	RGCMP RGMV3
Control Level Extraction	RGMV5
Matching Field Extraction	RGMV5 RGCMP
MFTST routine	RGMV3 RGCMP
IODs	See discussion of IODs
CHAIN (calculation)	RGCVB
LOKUP (calculation)	RGCMP RGMV3 RGNCP RGSI3 RGSI4
CALCULATIONS	RGADD RGCMP RGDIV RGMLT RGMVR RGMV3 RGMV4 RGNCP RGSI1 RGSI2 RGSI3 RGSI4 RGSUB RGTSZ
Table Load and Dump	RGMV1 RGMV2

Table 12. Routines that Call Library Subroutines

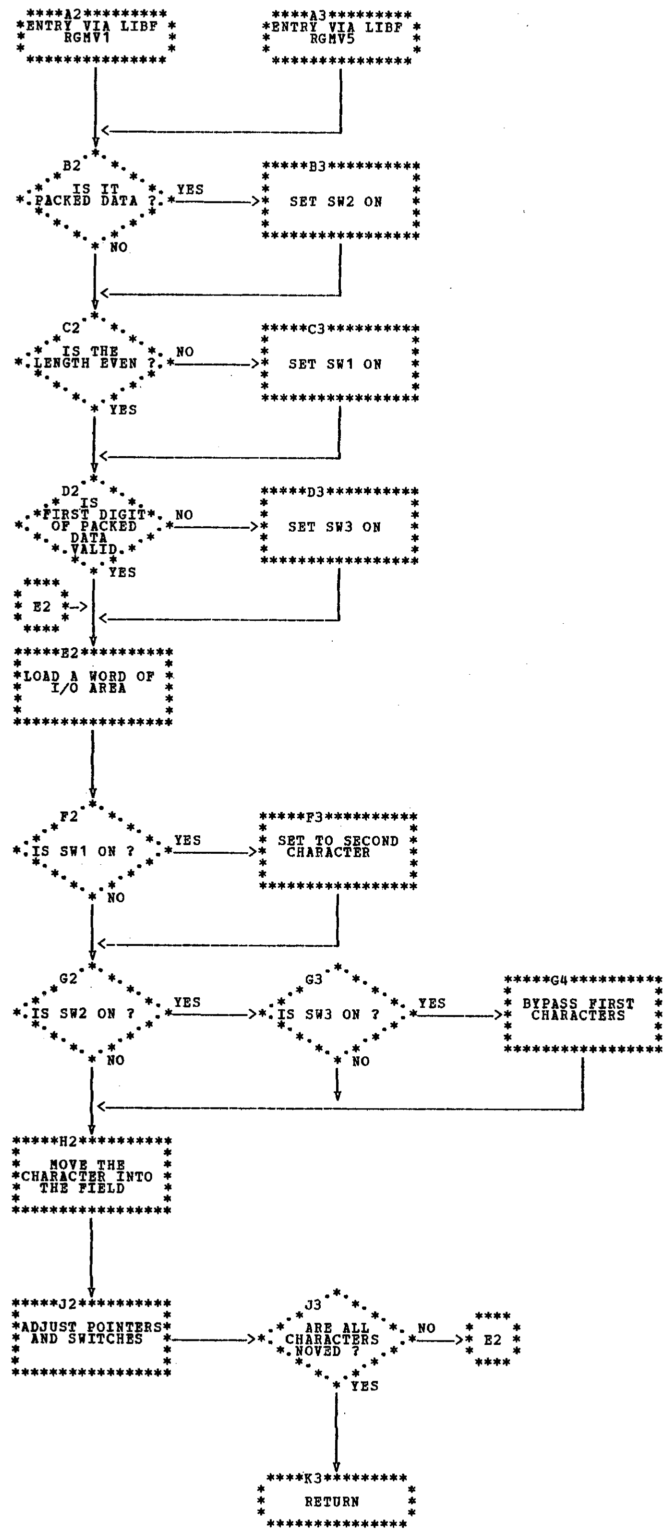


Chart MA. Move From I/O Buffer to Core Subroutine

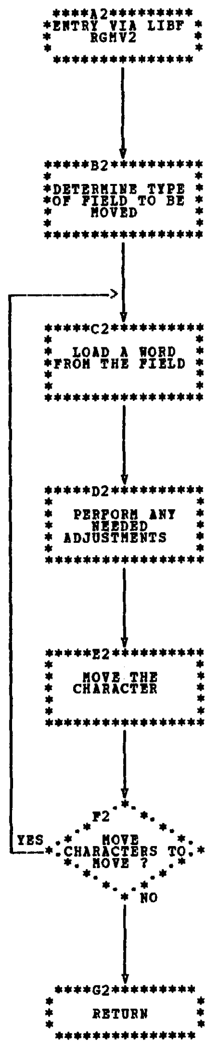


Chart MB. Move From Core to I/O Buffer Subroutine

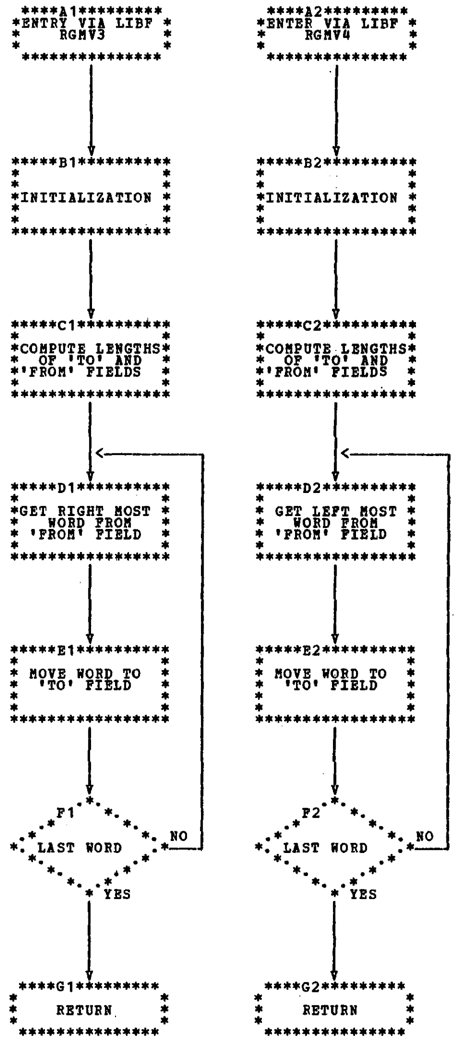


Chart MC. RPG MOVE and MOVEL Subroutines

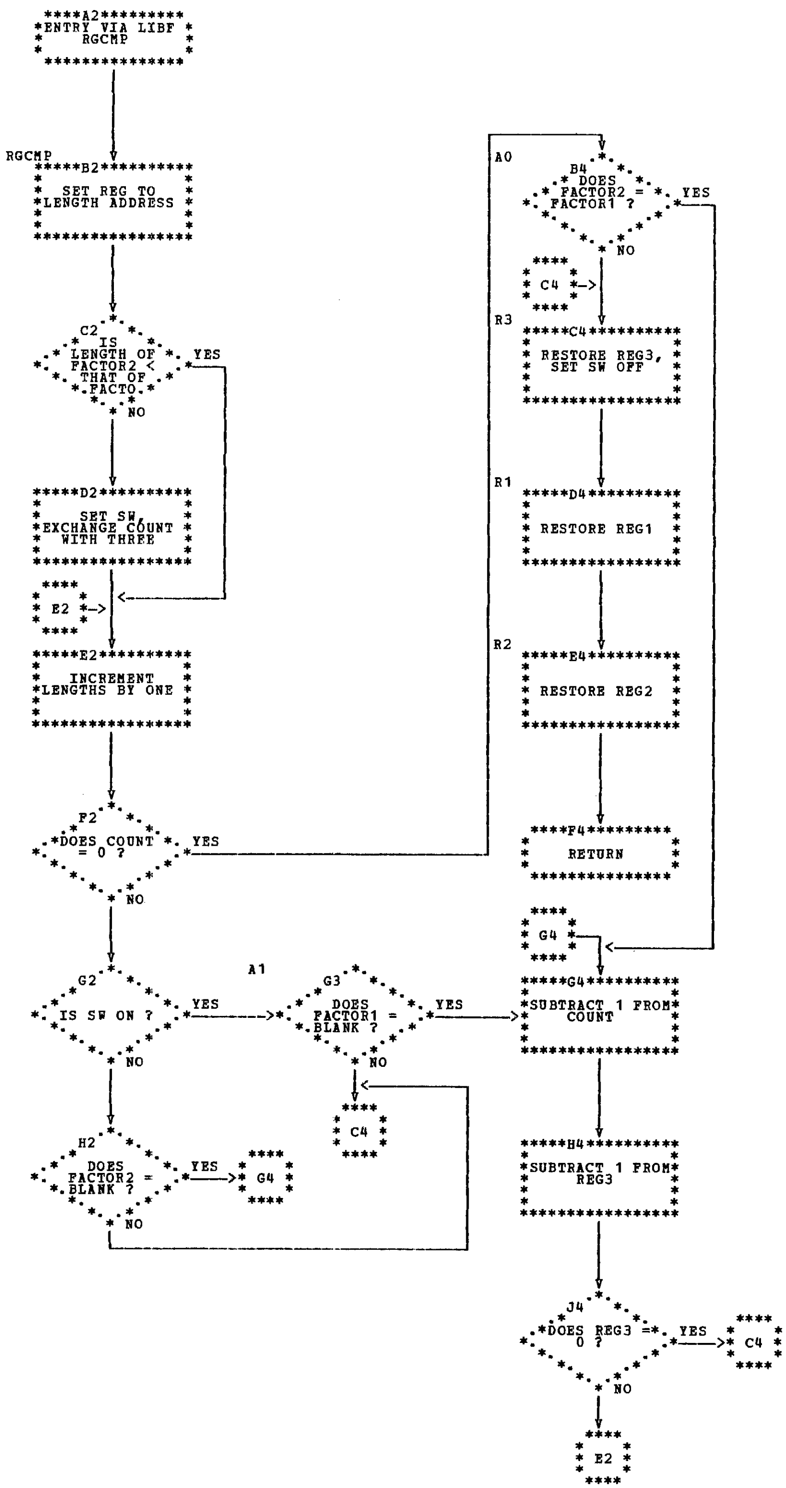


Chart MD. Alphameric Compare Subroutines

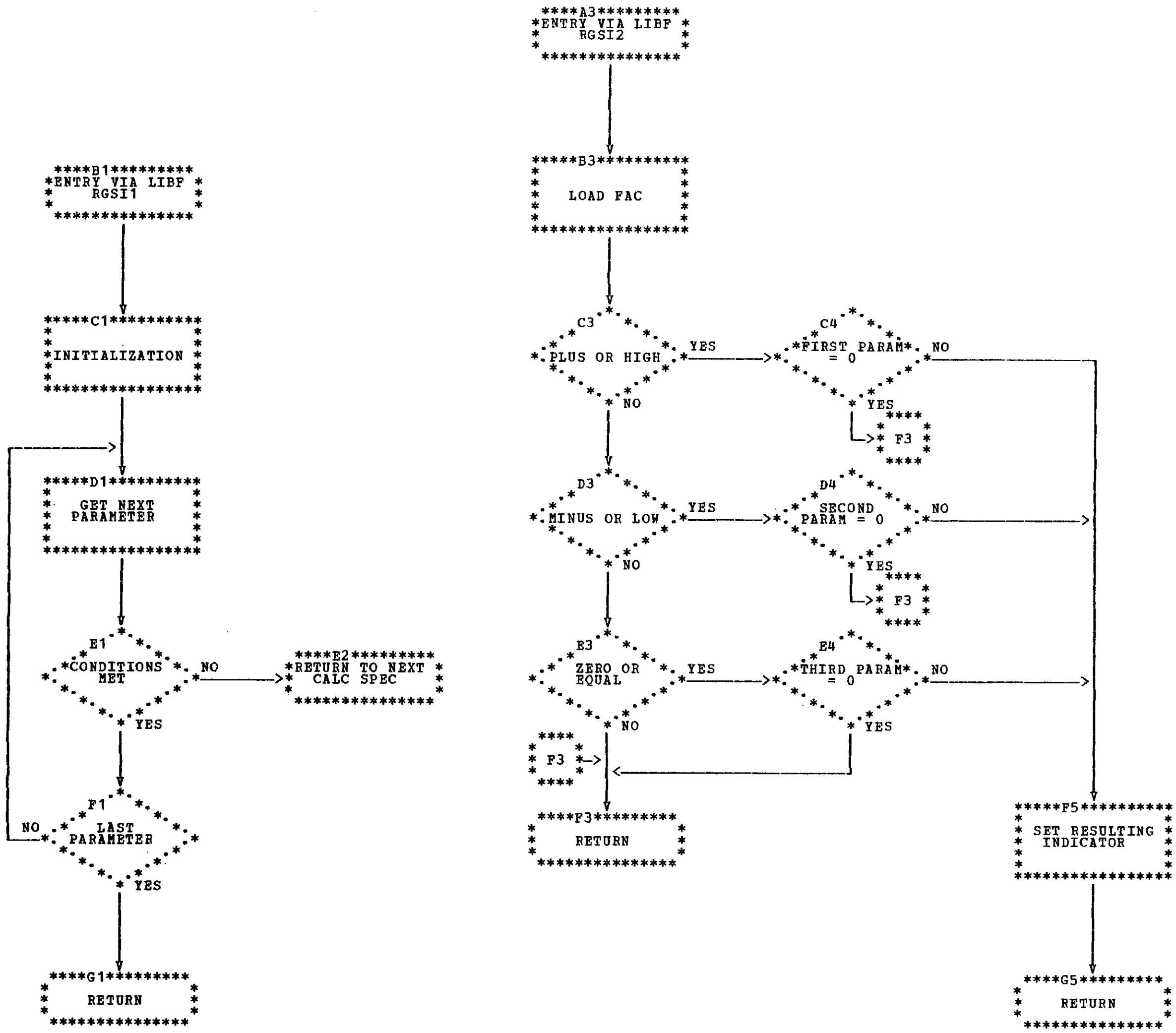


Chart ME. Test Indicators and Set Resulting Indicators Subroutines

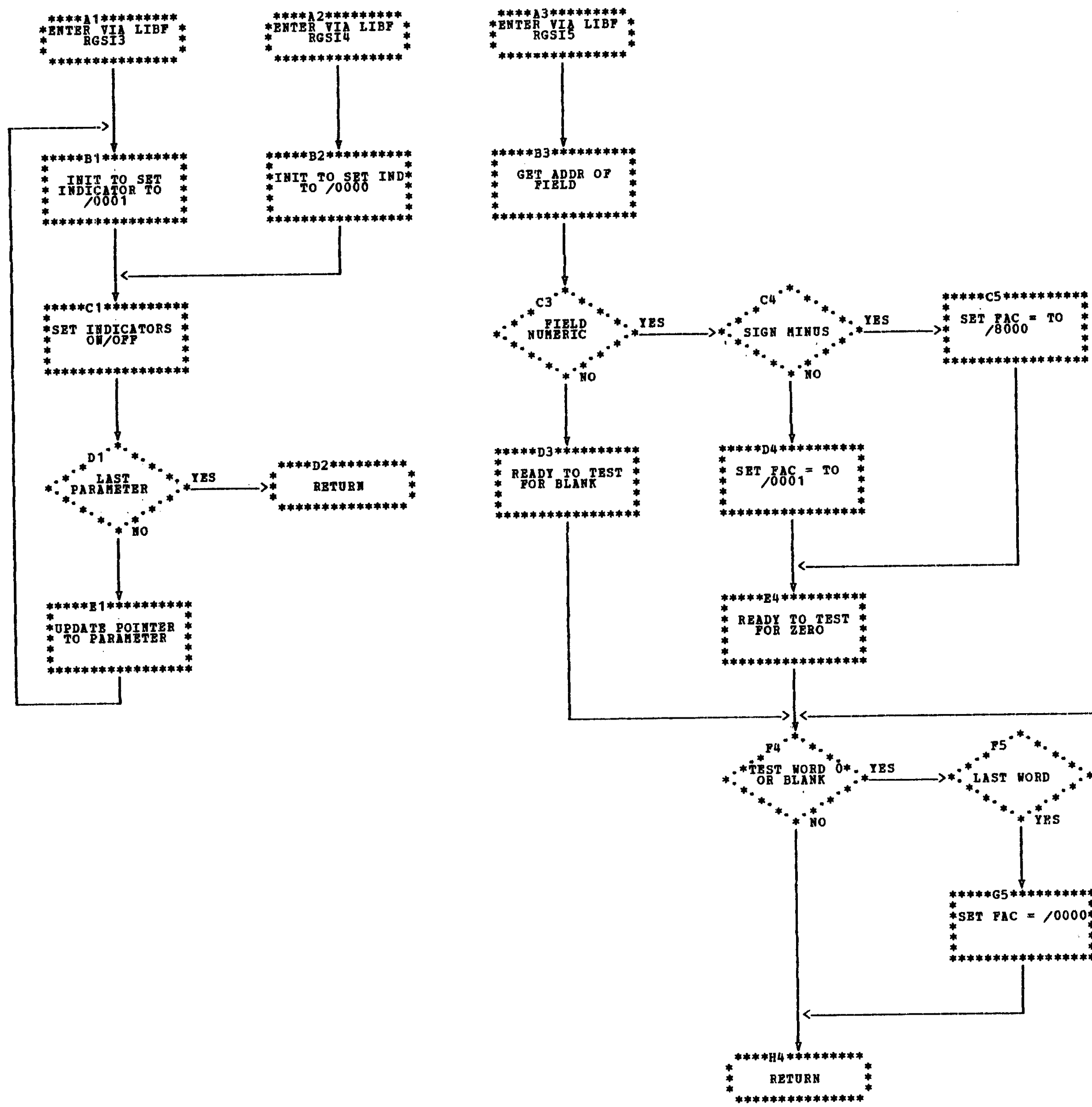


Chart MF. Set Indicators On or Off, and Test for Zero or Blank Subroutines

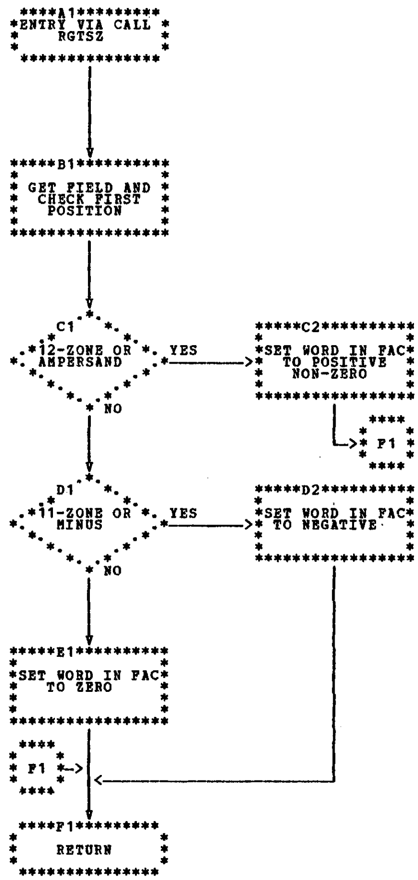


Chart MG. Test Zone Subroutine

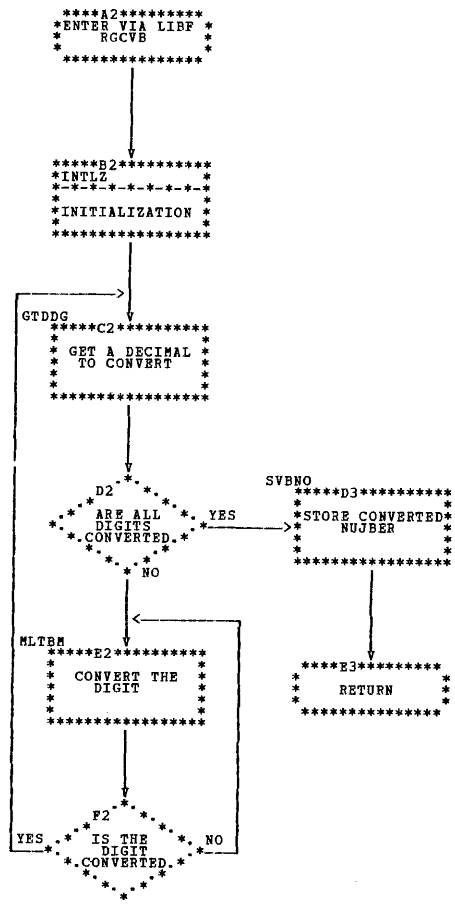


Chart MH. Record ID Conversion Subroutine

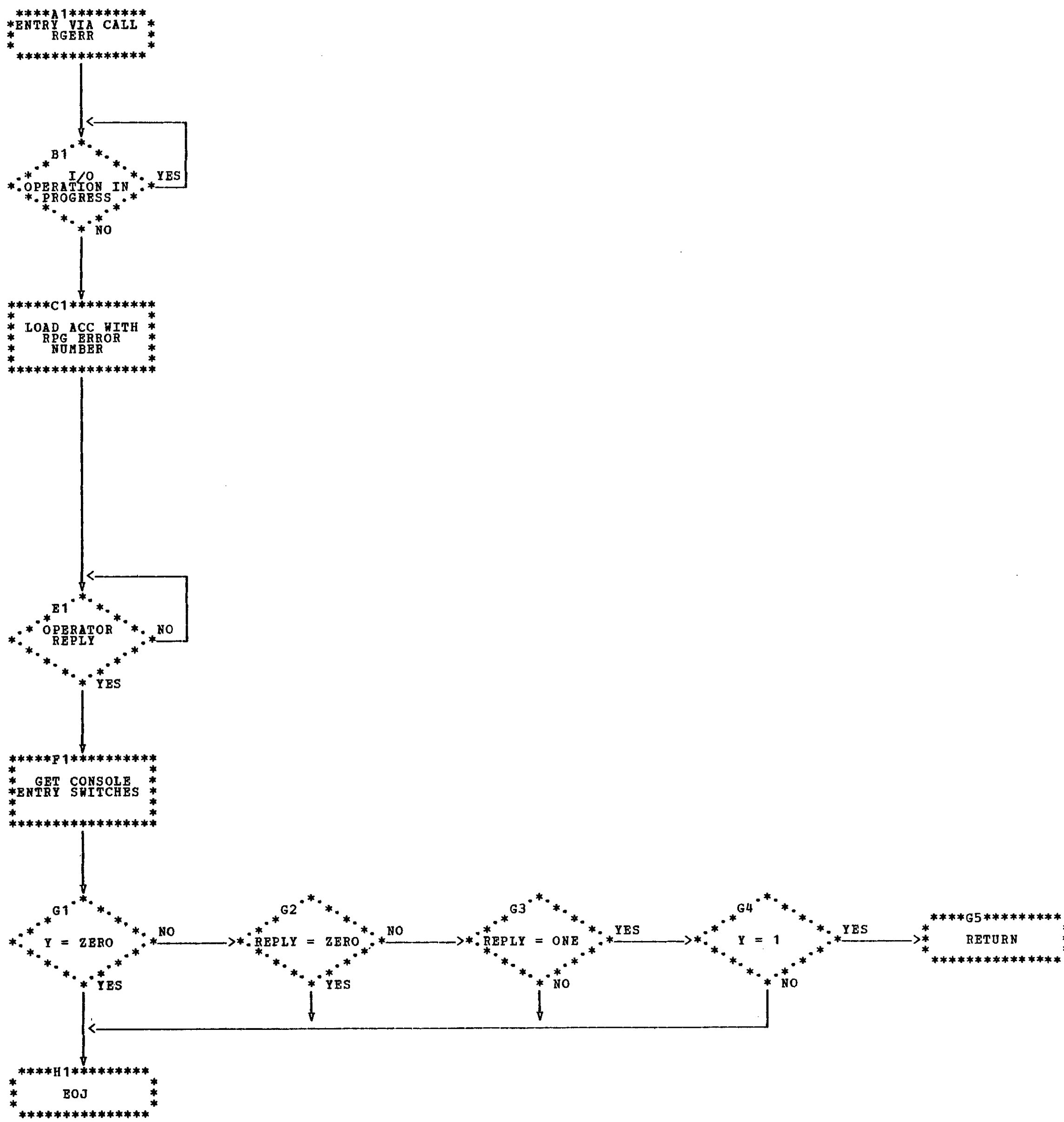


Chart MI. Object Time Error Subroutine

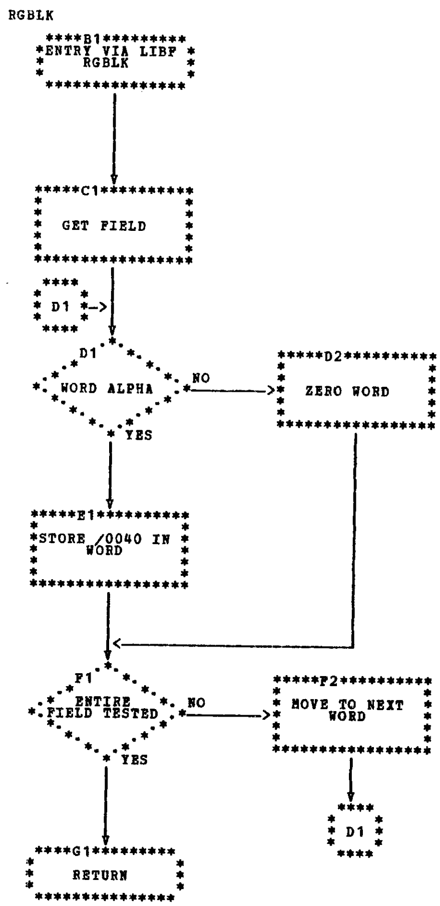


Chart MJ. Blank After Subroutine

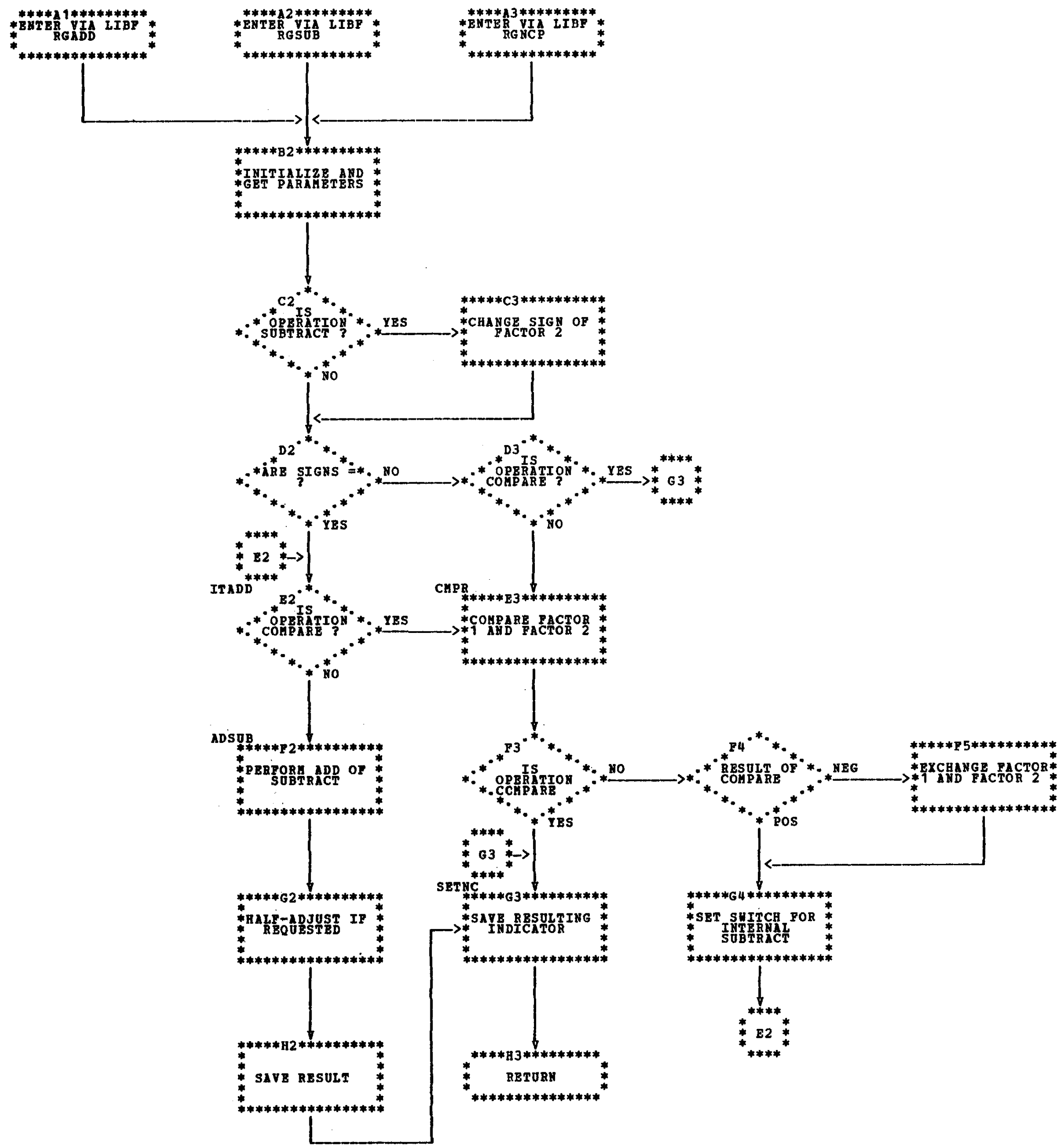


Chart MK. RPG Add, Subtract, Numeric Compare Subroutine

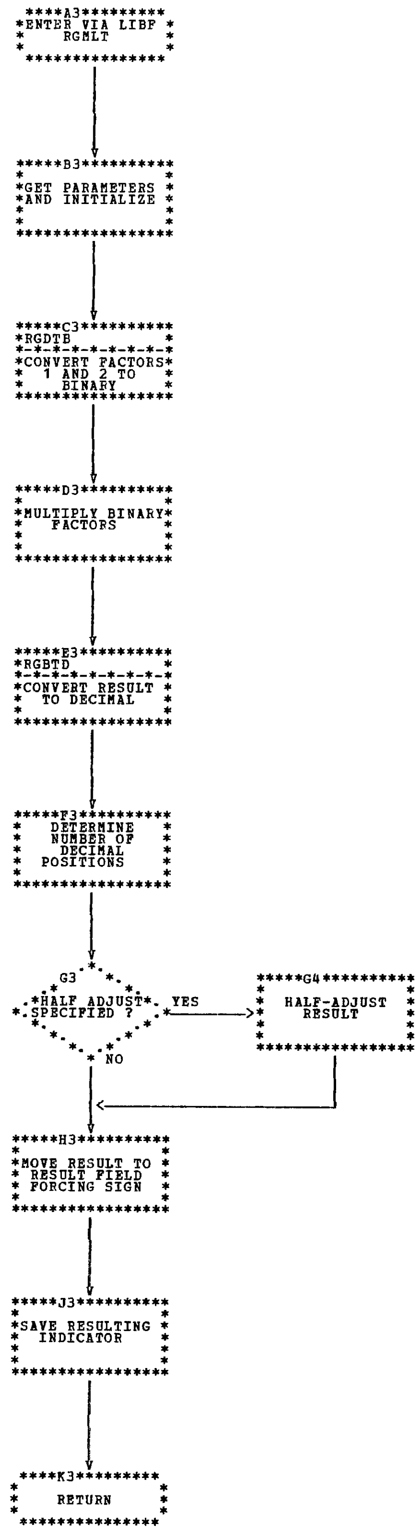


Chart ML. RPG Multiply Subroutine

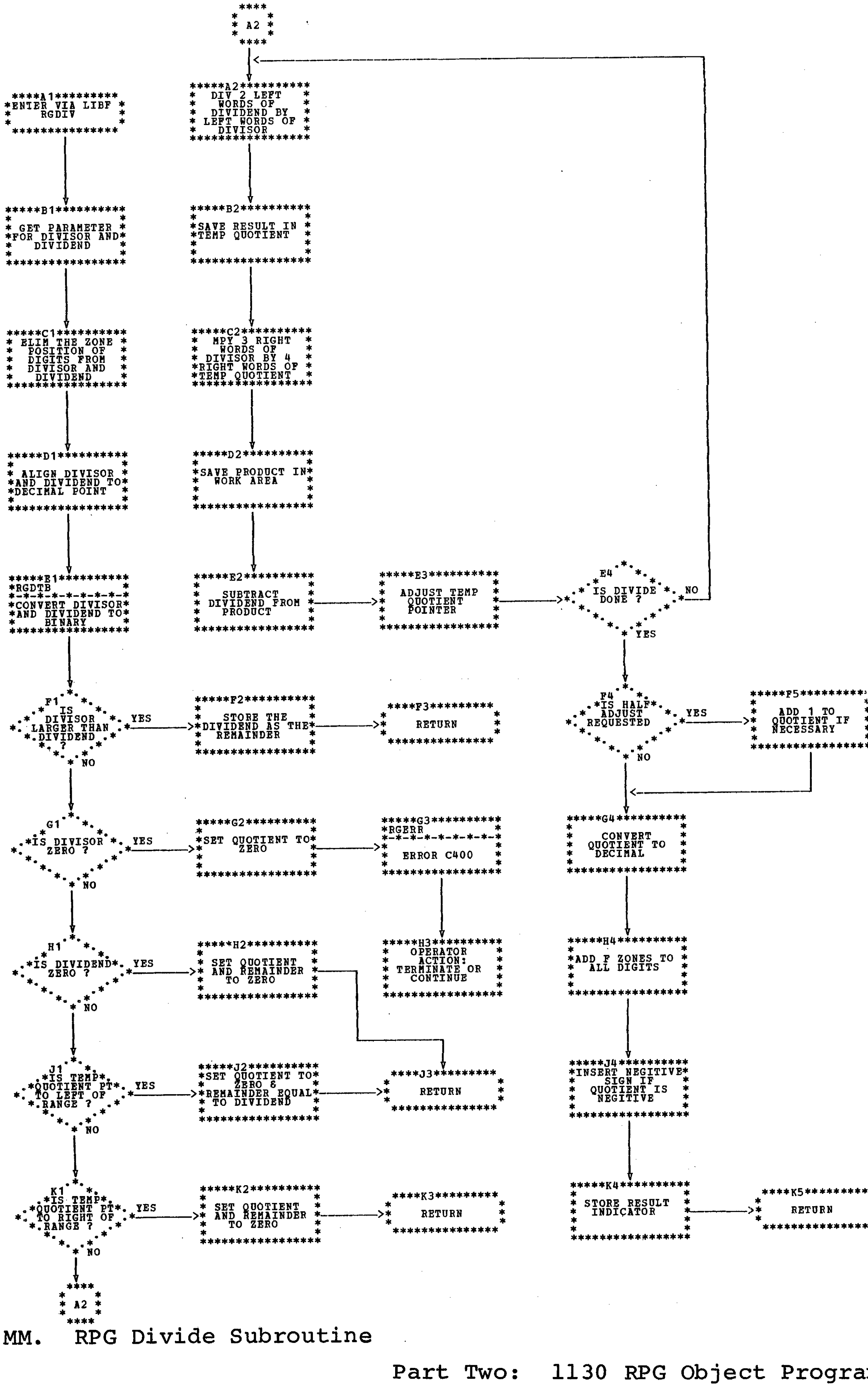


Chart MM. RPG Divide Subroutine

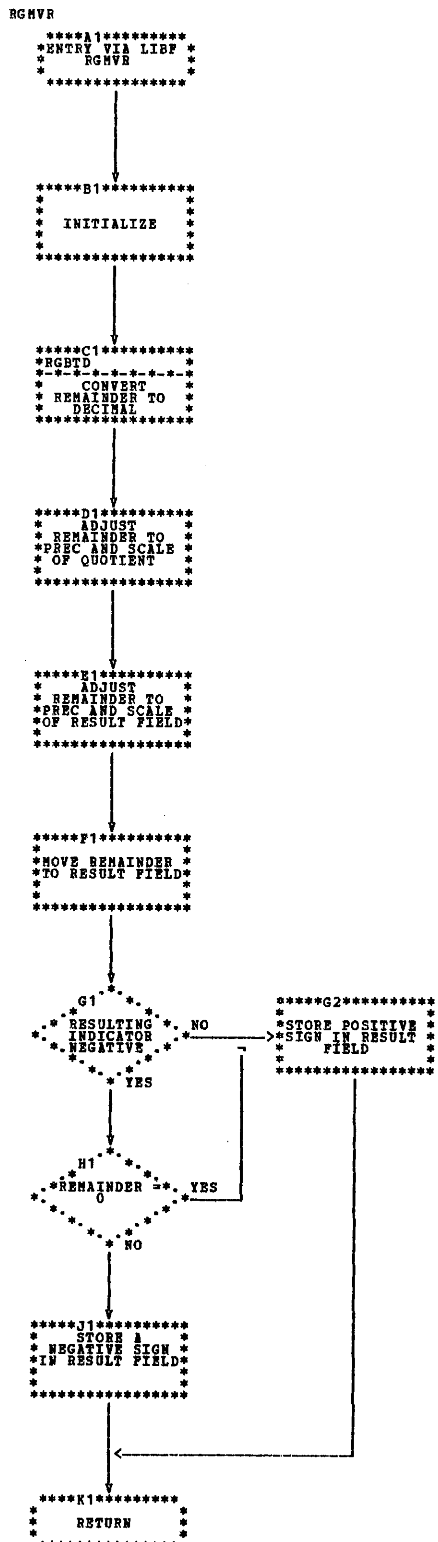
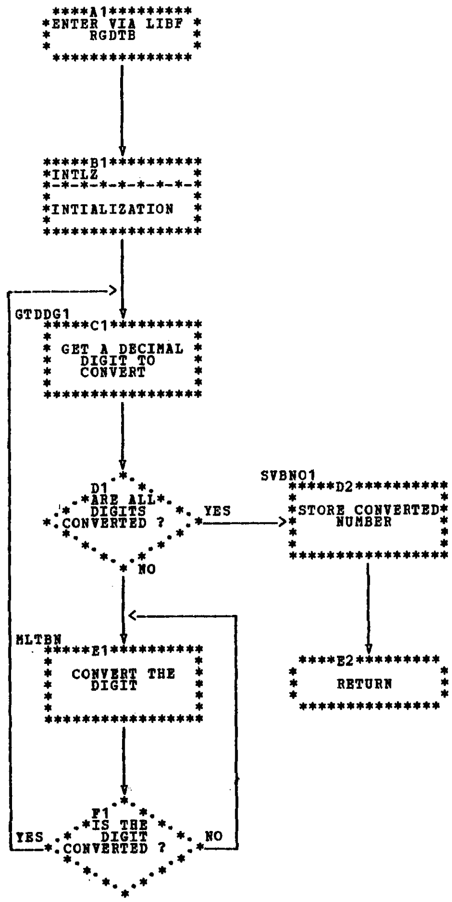


Chart MN. RPG Move Remainder Subroutine



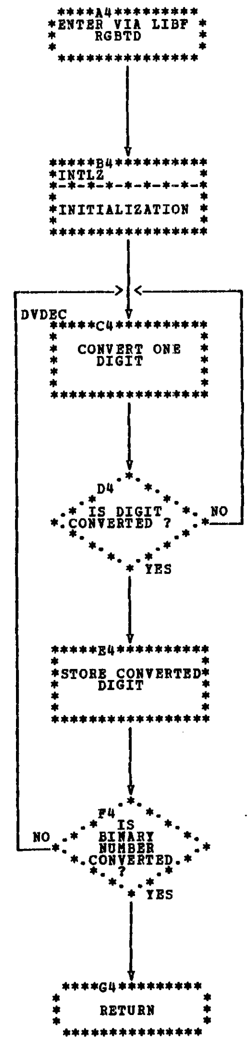


Chart MO. RPG Binary Conversion Subroutine

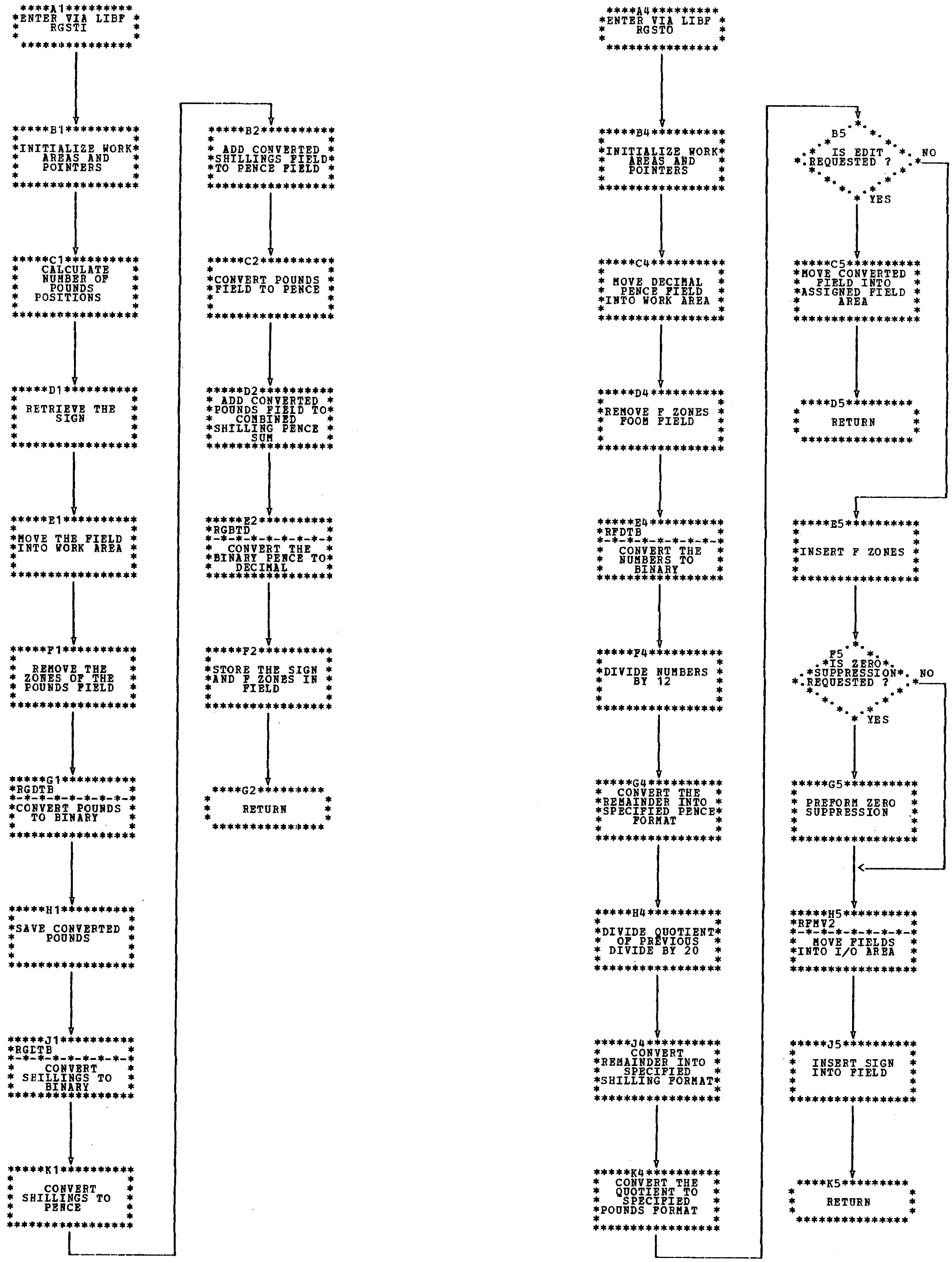


Chart MP. RPG Sterling Input and Sterling Output Conversion Subroutines

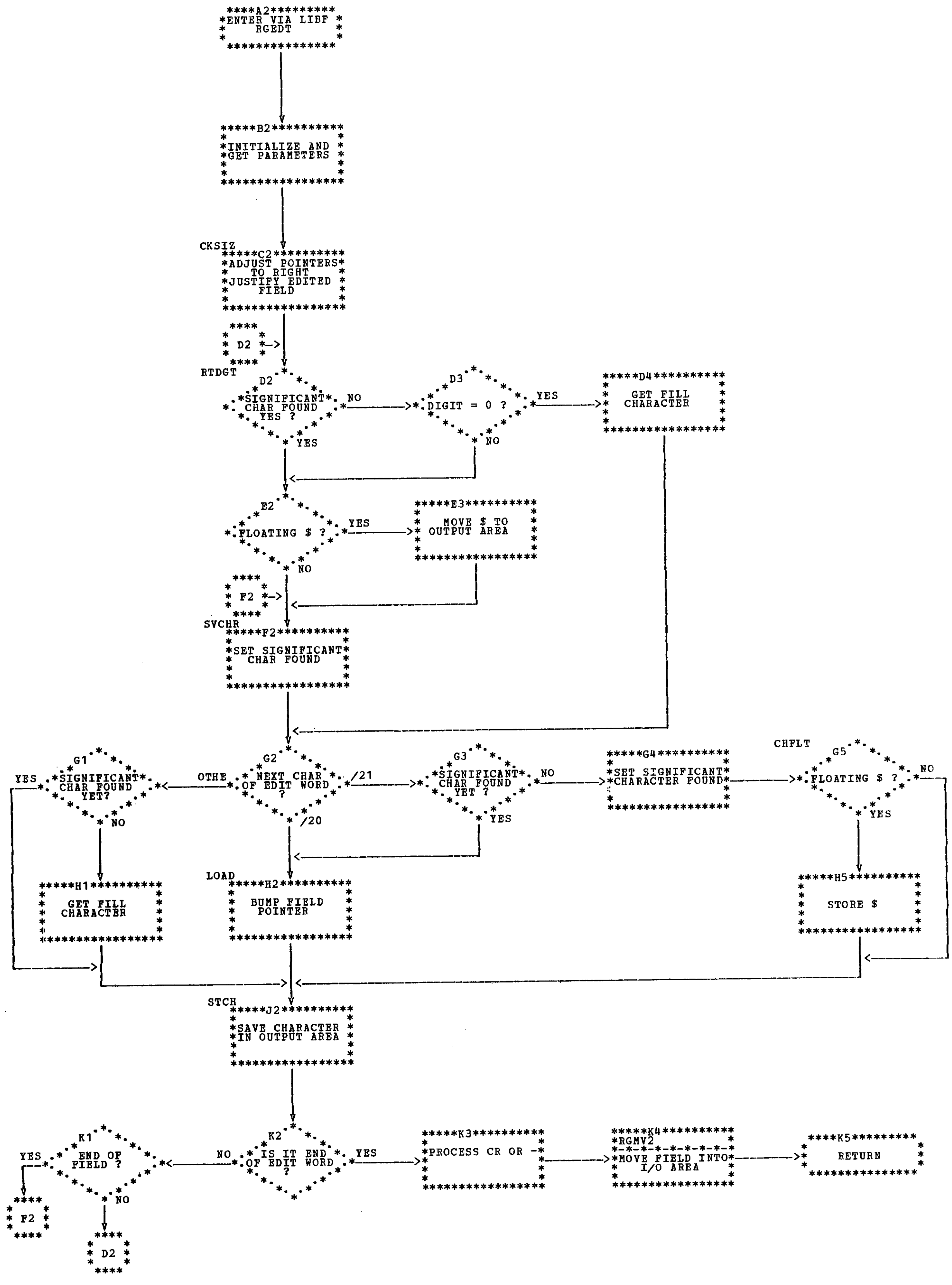


Chart MQ. RPG Edit Subroutine

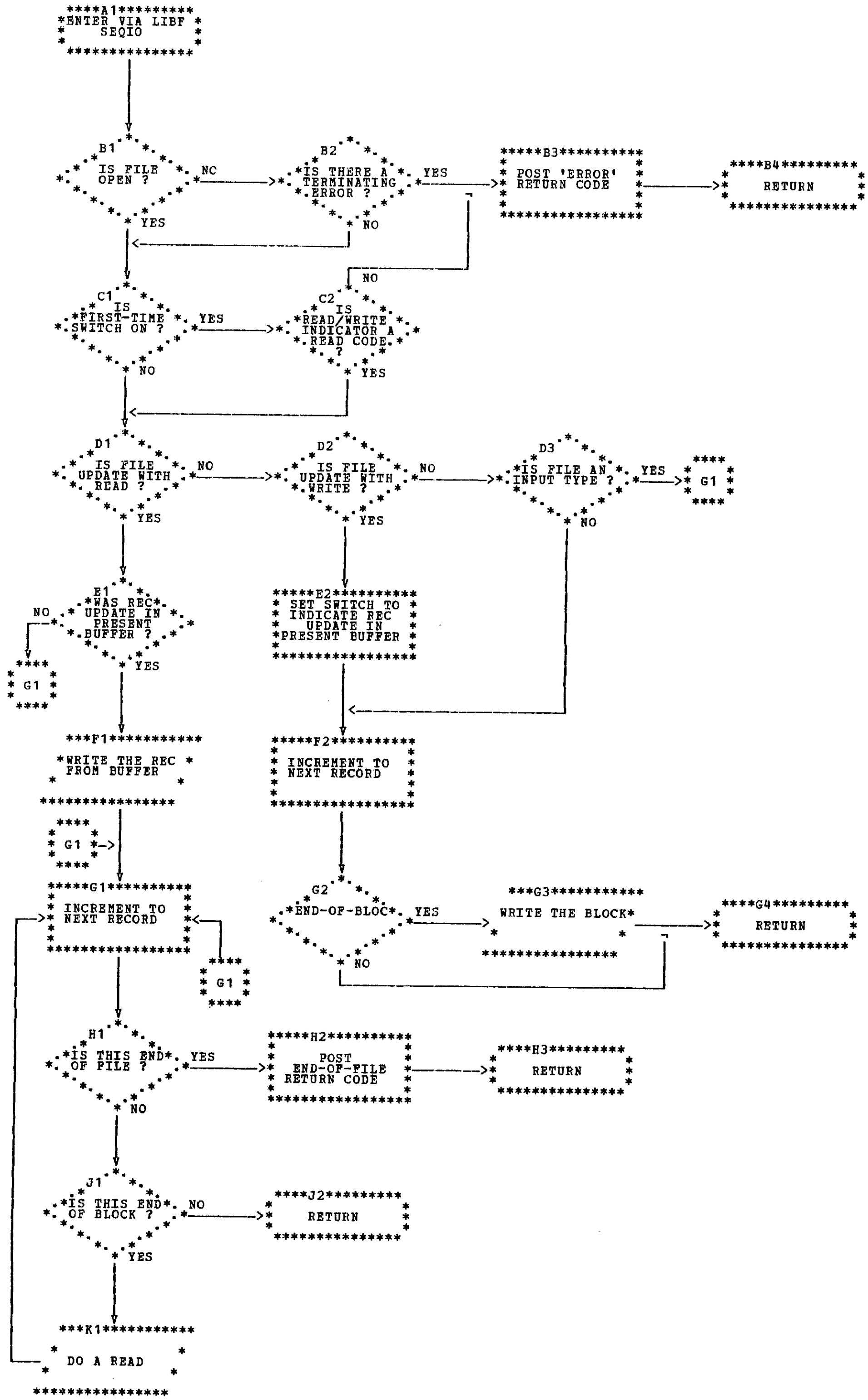


Chart MR. Sequential Disk Subroutine (Part 1 of 2)

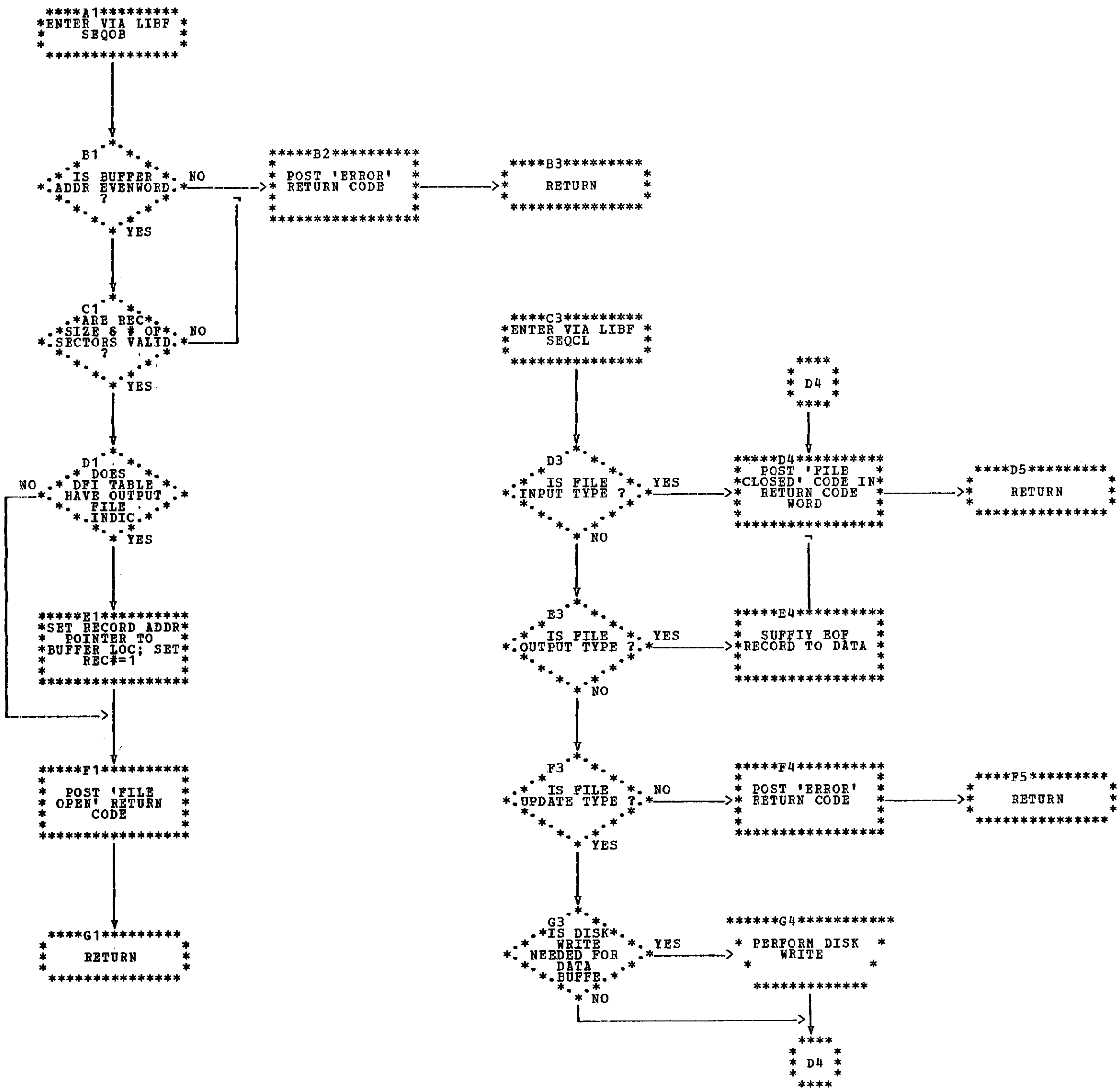


Chart MR. Sequential Disk Subroutine (Part 2 of 2)

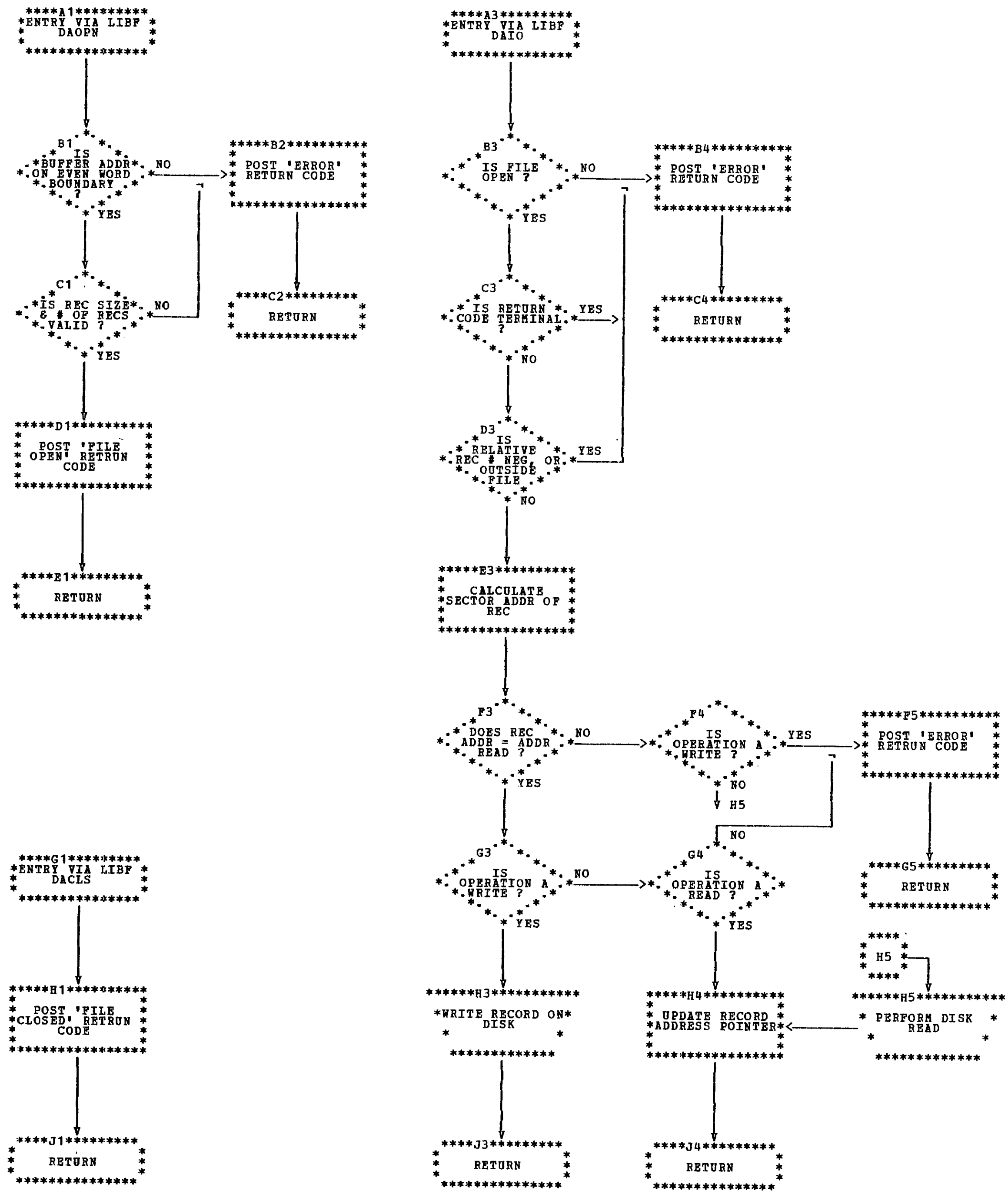


Chart MS. Direct Access Subroutine

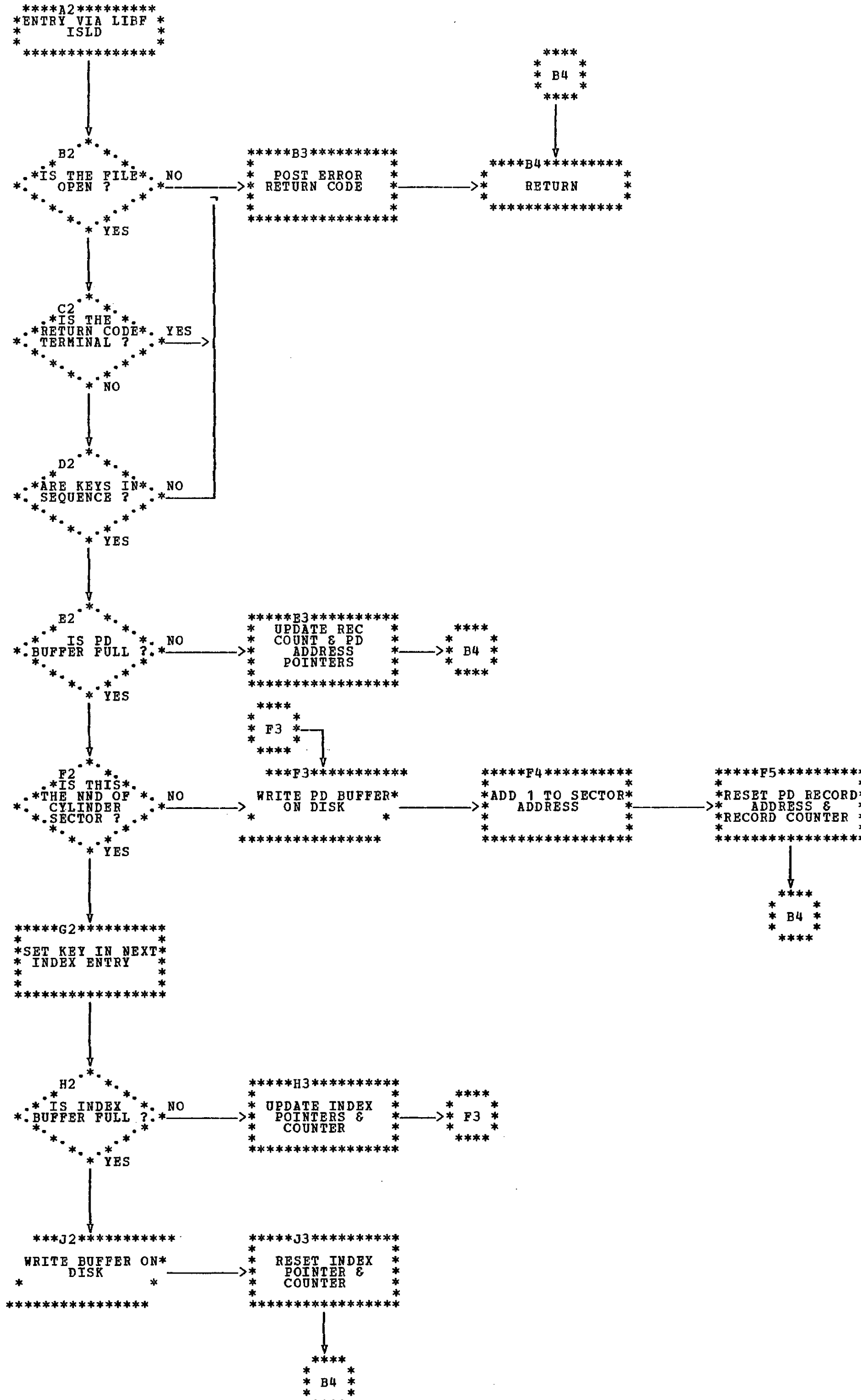


Chart MT. ISAM Load Subroutine (Part 1 of 2)

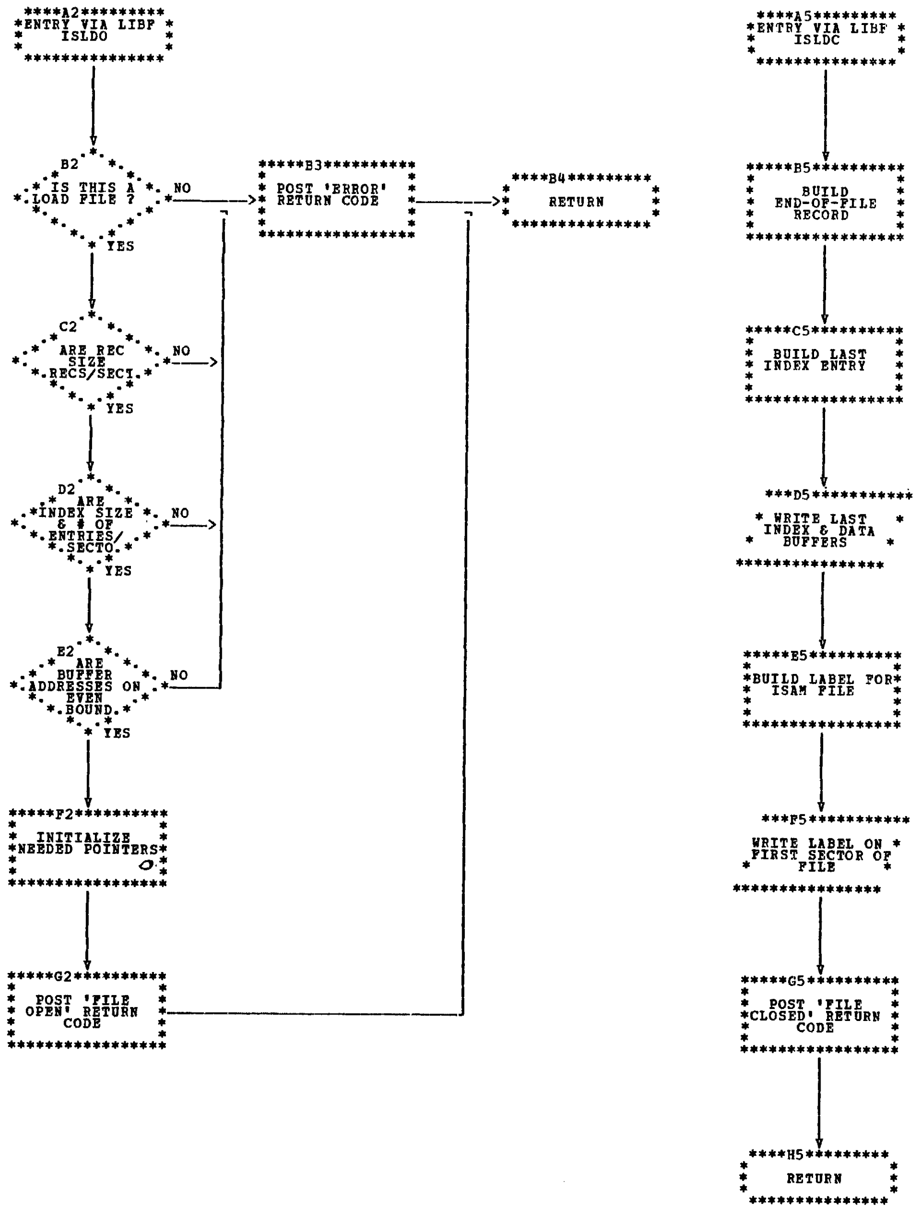


Chart MT. ISAM Load Subroutine (Part 2 of 2)

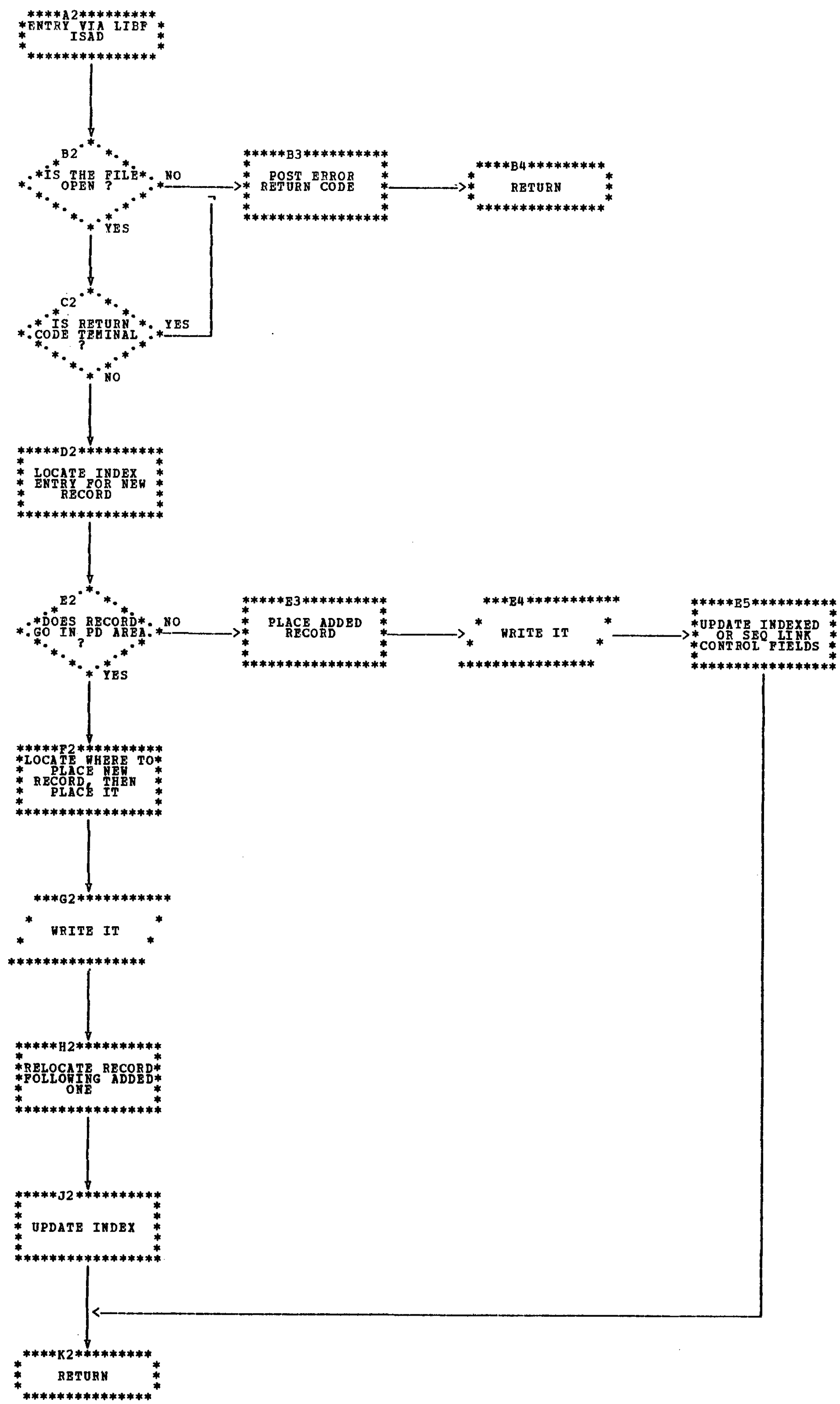


Chart MU. ISAM Add Subroutine (Part 1 of 2)

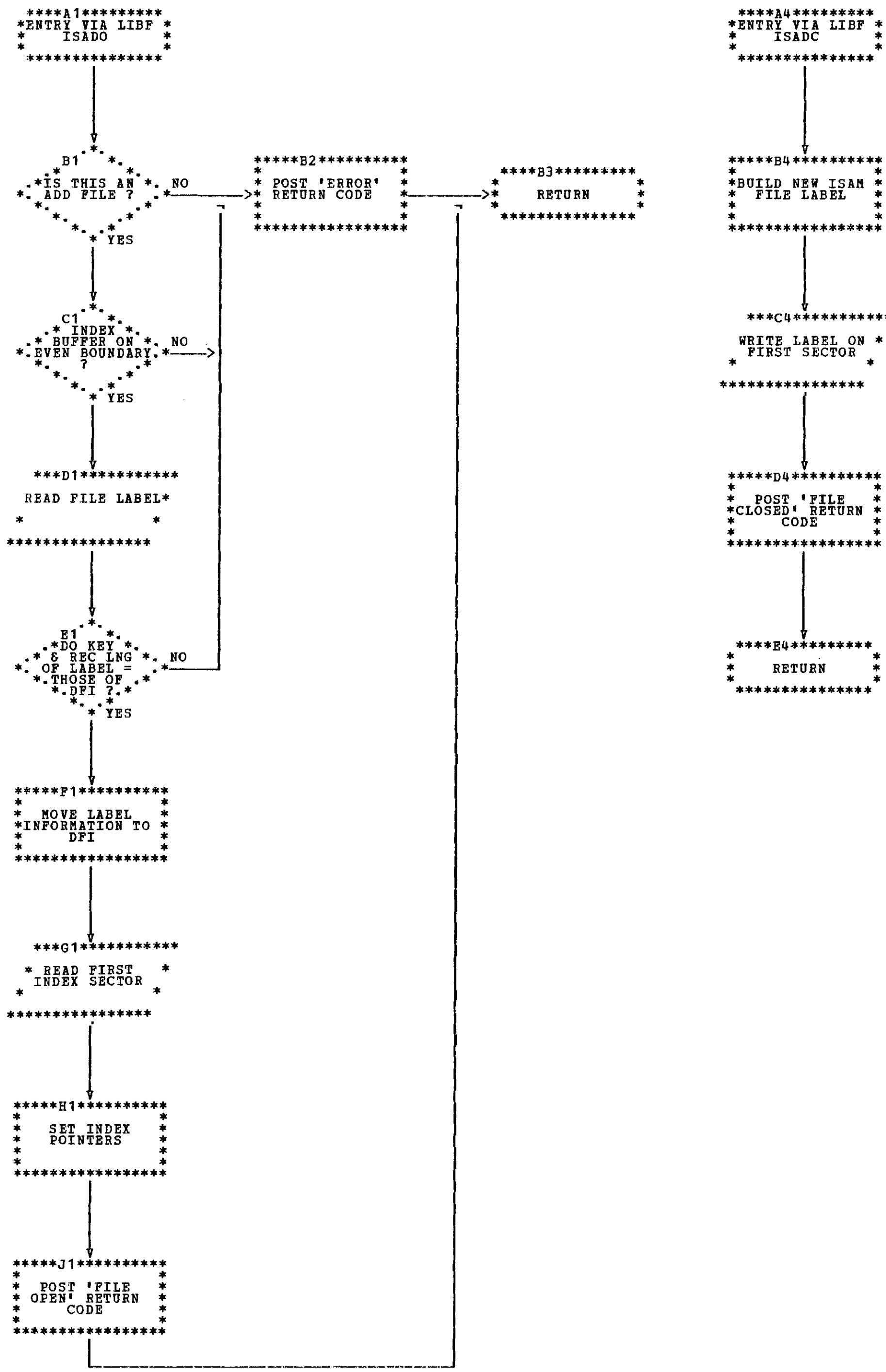


Chart MU. ISAM Add Subroutine (Part 2 of 2)

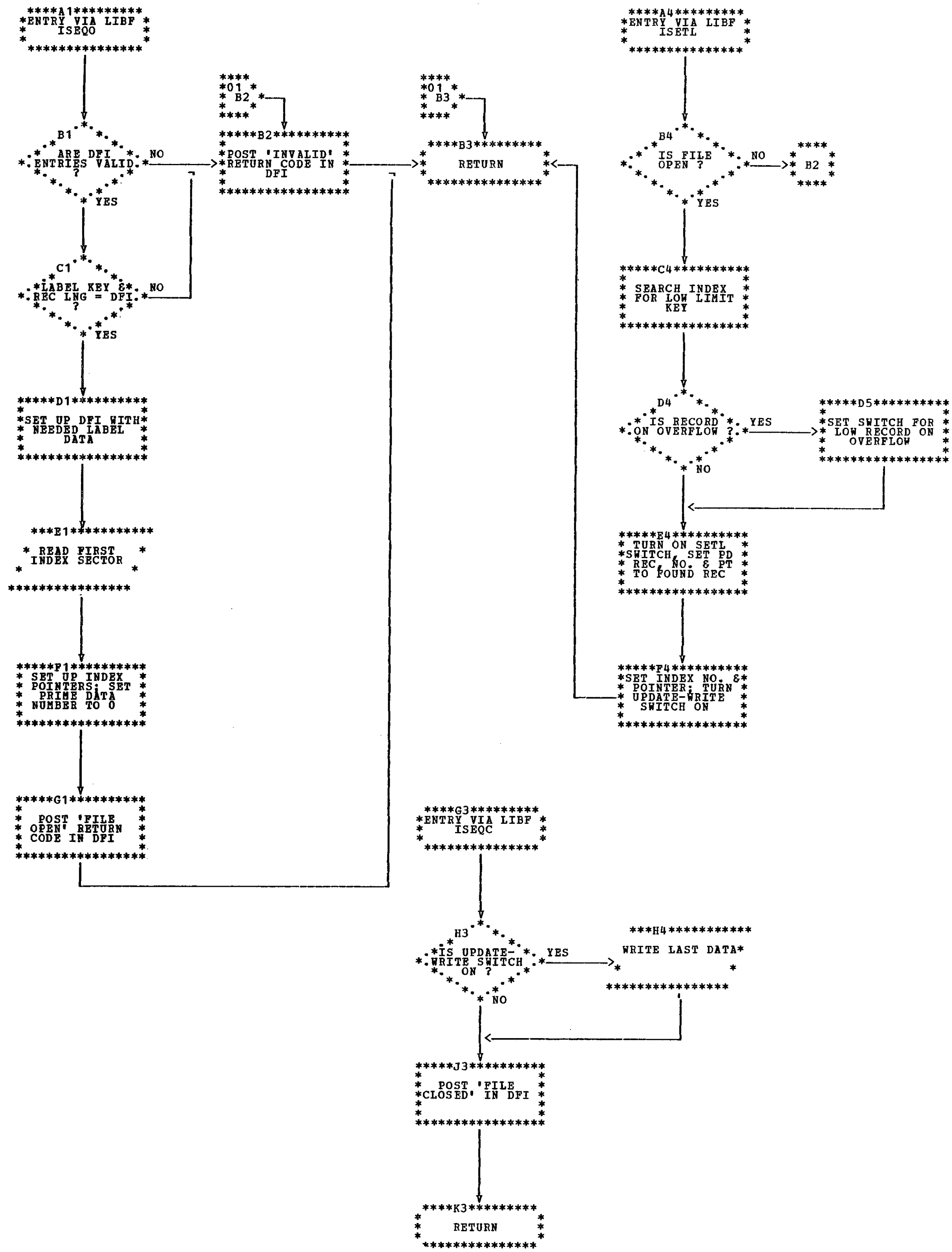


Chart MV. ISAM Sequential Subroutine (Part 1 of 2)

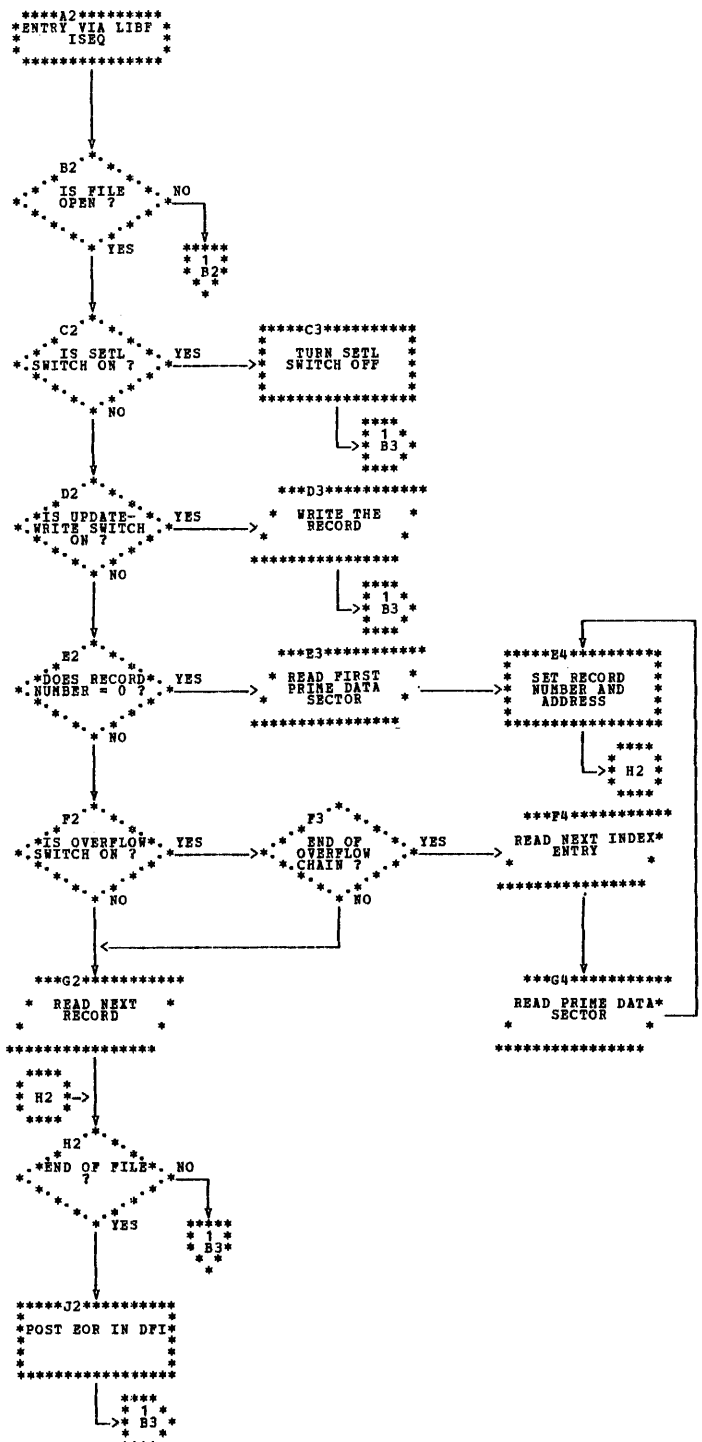


Chart MV. ISAM Sequential Subroutine (Part 2 of 2)

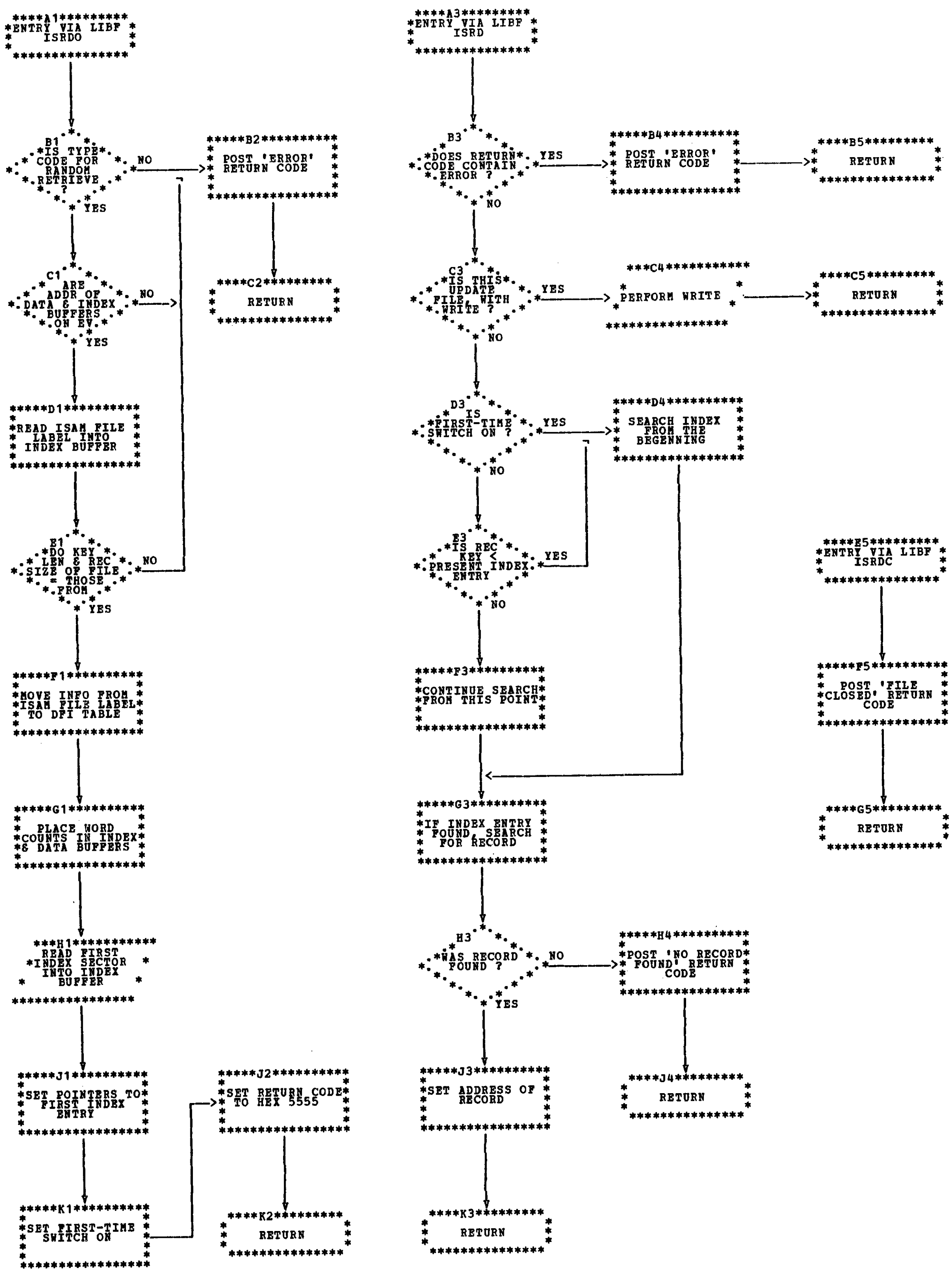


Chart MW. ISAM Random Subroutine

Move From I/O Buffer to Core (Chart MA)

Description of Operation

This subroutine has two entry points (RGMV1, RGMV5) and is used to move fields from the input I/O area to the fields assigned core storage area during execution of the RPG object program.

RGMV1: This routine begins by testing the first parameter for packed or unpacked format. If the format is packed, then it is necessary to determine from the I/O displacement (relative to the first character of the I/O area) whether the field starts on the first half of a word or the second half of a word. The address of the first position of the I/O area is in PR2. The routine then moves the field to the assigned fields area in an unpacked format.

RGMV5: This is another entry point into the routine at RGMV1 which will initialize RGMV1 with the field control word from the calling sequence rather than the TO field location. If the field is numeric, the sign position will be forced to /F by RGMV5 in the matching records hold area.

Calling Sequences

```
LIBF    RGMV1
DC      Zxxx where Z=0 if unpacked
                    =1 if packed
                 xxx=displacement of the
                        I/O area
DC      address of the I/O field
LIBF    RGMV5
DC      Zxxx (same values as above)
DC      address of the TO field location in
        the matching fields hold area
DC      field control word where
            bits 0-7 = length - 1
            bits 8-15= /00 if numeric
                     = /FF if alphameric
```

Move From Core to I/O Buffer (Chart MB)

Description of Operation

This subroutine has a single entry point (RGMV2) and is used to move fields from their assigned core locations into an output I/O area.

The routine begins by testing the first parameter for packed or unpacked output format. If the format is packed, it then determines from the I/O displacement (relative to the first character of the output I/O area) whether the field starts on the first half of a word or the second half of a word.

If bit 0 of the first parameter contains a 2, positive signs will be removed from the field as it is moved into the I/O area. If the bit is 3, zero suppression and sign removal is done on the field as it is moved.

Calling Sequence

```
LIBF    RGMV2
DC      Zxxx where Z = 0 for unpacked
                     = 1 for packed
                     = 2 for X edit code
                     = 3 for Z edit code
                   xxx = displacement of the
                         I/O area
DC      address of the From field
```

MOVE (Chart MC)

Description of Operation

This subroutine has a single entry point (RGMV3) and is used to perform the RPG MOVE operation for all fields. The field pointed at by the second parameter of the calling sequence is moved into the field pointed at by the first parameter and is right-justified within the field.

Calling Sequence

```
LIBF    RGMV3
DC      address of the TO field
DC      address of the From field
```

MOVEL (Chart MC)

Description of Operation

This subroutine has a single entry point RGMV4) and is used to perform RPG MOVEL operations for all fields. The field pointed at by the second parameter in the calling sequence is moved into the field pointed at by the first parameter and is left-justified within the field.

Calling Sequence

```
LIBF    RGMV4
DC      address of the To field
DC      address of the From field
```

Alphameric Compare (Chart MD)

Description of Operation

The alphameric compare subroutine has a single entry point (RGCMP) and is used to perform the compare operation for alphameric fields. The fields in Factor 1 and Factor 2 are compared for length first: the shorter will be extended with blanks. Then, the values are compared with Word 123 + XR3 in the FAC being set as follows:

Condition	Setting
Factor 1 > Factor 2	Positive non-zero
Factor 1 < Factor 2	Negative
Factor 1 = Factor 2	Zero

Calling Sequence

```
LIBF    RGCMP
DC      address of Factor 1
DC      address of Factor 2
```

Test Indicators (Chart ME)

Description of Function

The test indicators subroutine has one entry point (RGSI1) and is used to test the condition of the control-level indicator and up to three more indicators on a calculation operation.

Description of Operation

The control-level and up to three additional indicators are tested on a calculation operation. Bit 0 of each indicator will be set to 1 if the test is for a not condition. If conditions are met, return will be to the word following the last parameter in the calling sequence. If conditions are not met, return will be to the next calculation specification.

Calling Sequence

```
LIBF    RGSI1
DC      Bits 0-2 = the number of parameters,
        including this parameter.
        Bits 3-15 = displacement from this
        DC to next calculation operation.

DC      address of control-   )
        level indicator       ) from 1 to 4
DC      address of indicator X) indicator
DC      address of indicator Y) addresses.
DC      address of indicator Z)
```

Set Resulting Indicators (Chart ME)

Description of Function

The set resulting indicators subroutine has one entry point (RGSI2) and sets resulting indicators dependent upon the result of an arithmetic or compare or TESTZ operation.

Description of Operation

If the designated operation results in a + or high condition and the first parameter contains an address, the indicator is set on (/0001). If the result is - or low and the second parameter contains an address, indicator Y is set on. If the result is 0 or equal, and the third parameter contains an address, indicator Z is set on. If there is no indicator specified, the associated parameter will contain /0000.

Calling Sequence

```
LIBF    RGSI2
DC      address of indicator X
        (+, high, 12-zone, or X'C0')
DC      address of indicator Y
        (-, low, 11-zone, or X'D0')
DC      address of indicator Z
        (0, =, all other zones)
```

Set Indicators On or Off (Chart MF)

Description of Operation

The set indicators on (SETON) entry point (RGSI3) may be used to set on (/0001) as many as three indicators. The last parameter present will have bit 0=1 to stop the operation and return to the next instruction after the last parameter in the calling sequence.

Calling Sequence

```
LIBF    RGSI3                   ) One to three
DC      address of indicator X  ) of these pa-
DC      address of indicator Y  ) rameters may
DC      address of indicator Z  ) be present.
                                ) Last one will
                                  have bit 0=1.
```

The set indicators off entry point (RGSI4) is used for SETOFF and TESTZ operations, and for clearing indicators before arithmetic or compare operations. Up to three indicators may be cleared and the last parameter in the calling sequence will have bit 0=1 to stop the operation and return will be made to the instruction following the last parameter.

Calling Sequence

```
LIBF    RGSI4        )
DC      address of   ) One to three of these
        indicator X  ) parameters may be
DC      address of   ) present.  Last one
        indicator Y  ) will have bit 0=1.
DC      address of   )
        indicator Z  )
```

Test for Zero or Blank (Chart MF)

Description of Operation

The test for +, -, 0 or blank subroutine (entry point RGSI5) tests a field for these conditions and sets a word (XR3+123) in the FAC as follows:

- Numeric field

plus	/0001
minus	/8000
zero	/0000

- Alphameric field

blank	/0000
non-blank	/000F

The +, -, 0 or blank indicator will be set by linking to RGSI2 using the FAC.

Calling Sequence

```
LIBF    RGSI5
DC      address of the field
```

Test Zone (Chart MG)

Description of Function

The test zone subroutine has a single entry point (RGTSZ) and is used to test the zone of an alphameric field.

Description of Operation

The zone of the leftmost position of the field supplied is tested and a word in the FAC (Index register 3+123) is set as follows:

- If zone is a 12-zone or the same zone as a letter A or an ampersand (&), set word to non-zero position.
- If zone is an 11-zone or the same zone as a letter J or a minus (-), set word to negative.
- If zone is any other zone, set word to zero.

Calling Sequence

```
CALL    RGTSZ
DC      address of the field to be tested
```

Record ID Conversion (Chart MH)

Description of Operation

The record ID conversion subroutine has a single entry point (RGCVB) and is used to convert the record ID supplied by the RAF field or chaining field into a two-word binary number for direct access processing.

Calling Sequence

```
LIBF    RGCVB
DC      address of the field to be converted
DC      address of the two-word hold area
        for the converted result
```

Object Time Error (Chart MI)

Description of Function

The object time error subroutine has a single entry point (RGERR) and is used to give the operator an opportunity to terminate or continue if an error occurs.

Description of Operation

An error number is loaded into the accumulator and a wait state is entered at $PRET. The operator must reply with /0000 to terminate or /0001 to bypass record and continue. If the decision is terminate, the EOJ address is calculated in the Fixed Driver and a branch is made to that address. The bypass and continue option returns to the word following the calling sequence.

Calling Sequence

```
CALL    RGERR
DC      YXXX   where Y=0 if error is termin-
               al, or 1 if error may be by-
               passed to continue the job.
               XXX = a three digit error
                     number.
```

Blank After (Chart MJ)

Description of Function

The blank after subroutine has one entry point (RGBLK) and is used to set output fields to blank if the field is alphameric, or zero if the field is numeric.

Description of Operation

The length of the field and its status (numeric or alphameric) is determined. If the field is numeric, all words are set to /0000. If the field is alphameric, all words are set to /0040.

Calling Sequence

```
LIBF   RGBLK
DC     address of the field to be zeroed
       or blanked
```

<u>Add, Subtract and Numeric Compare (Chart MK)</u>

Description of Functions

This subroutine has three entry points, RGADD, RGSUB, and RGNCP.

<u>RGADD</u>: This entry point is for the add function. Add will take the input data from factor 1 and factor 2, remove the zones, and align the two fields to their decimal points. If the signs of factor 1 and factor 2 are the same, the two factors will be added to form the result, with the sign of the result being the same as the factors.

If the signs are not the same, the two factors are compared to see which factor is larger. If factor 1 is larger, factor 2 will be subtracted from factor 1 and the result will have the sign of factor 1. If the factor 2 is larger, factor 1 will be subtracted from factor 2 and the result will have the sign of factor 2.

Upon completion of the operation, the result will be moved to the output field specified by the calling sequence. A zone of 'F' will be attached to all digits unless the result is negative, in which case a sign of 'D' will be inserted.

<u>RGSUB</u>: This entry point is for the subtract function. For subtract, the sign of factor 2 is changed and processing continues as in the add function.

<u>RGNCP</u>: This entry point is for the numeric compare function. Numeric compare takes the input data from factor 1 and factor 2, removes the zones, and aligns the two fields to their decimal points. The two fields are then compared. Upon completion of the compare, an indicator is set to indicate the result of the compare and a return is made to the calling routine.

Description of Operation

When the subroutine is entered, a switch is set to indicate what type of operation is to be performed (add, sub or numeric compare) after which an initialization routine is entered to retrieve the calling sequence and set up work areas within the program to correspond with the entries in the calling sequence. The work areas for factor 1 and factor 2 are set to zero in the initialization. (If the operation is numeric compare, the result field is not initialized.) After initialization, the contents of factor 1 and factor 2 are moved to the work area. The zone bits are removed at this time, leaving one digit per word right justified.

The signs are then compared to decide how calculations are to be performed (if the operation is subtract, the sign of factor 2 has been changed). If the signs are the same and the operation is add or subtract, an add operation is performed. If the signs are the same and the operation is compare, a compare operation is performed to determine which value is larger. If the signs are different and the operation is add or subtract, a subtract operation is performed. If the signs are different and the operation is compare, the signs are again checked and the one with the positive sign is determined to be larger. The resulting indicator is stored and a return is made to the calling routine.

To perform a compare operation, decimal points must first be aligned. The decimal points are aligned by adjusting the index pointers to the data fields (factor 1 and factor 2). The compare is performed by subtracting factor 2 from factor 1. If the result is not zero, there is an unequal compare. If zero, the compare is equal.

Upon completion of the compare routine, the operation type is checked again. If the operation is compare, the correct resulting indicator is stored and a return is made to the calling routine. If the operation is add or subtract and factor 1 is equal to or larger than factor 2, the subtraction routine is entered. If factor 1 is less than factor 2, the two fields are switched and the subtract routine is entered.

If the add or subtract routine is to be performed, decimal points must first be aligned as they were for numeric compare. The add routine is performed by adding factor 1 to factor 2. If the result is greater than ten, add six to the result, strip the 12 high-order bits and store as the result, and add one to the next digit. This loop is continued until the result field is satisfied.

The subtract routine is performed by subtracting factor 2 from factor 1. If the result is less than zero, add ten to the result and subtract one from the next highest digit of factor 1. This loop is also continued until the result field is satisfied.

If half-adjust is specified, it is performed by adding the low-order digit plus one or the result to itself. If the result is greater than nine, add one to the low-order digit of the result. The result data is then moved to the result field specified by the calling sequence. A zone of 'F' will be attached to all digits unless the result is negative, in which case a sign of 'D' will be inserted.

The final step before returning to the calling routine is to store an indicator in word XR3+123 as follows, to indicate the sign of the result:

plus - /ØØØ1
minus - /8ØØØ
zero - /ØØØØ

Work Areas Used:

FACT1 - This 24 word work area is used to contain factor 1 data in the format 000D where D is any decimal digit. This work area can also contain the result of an add or subtract operation.

FACT2 - This 23 word work area is used to contain factor 2 data in the format 000D where D is any decimal digit. This work area can also contain the result of an add or subtract operation.

Calling Sequences

```
LIBF   RGADD
DC     Address of Factor 1
DC     Address of Factor 2
DC     Address of Result Field
DC     Half-Adjust Indicator

LIBF   RGSUB
DC     Address of Factor 1
DC     Address of Factor 2
DC     Address of Result Field
DC     Half-Adjust Indicator

LIBF   RGNCP
DC     Address of Factor 1
DC     Address of Factor 2
```

Multiply (Chart ML)

Description of Function

The multiply subroutine has a single entry point (RGMLT). Multiply takes two fields with a maximum length of fourteen decimal digits and converts each word to an unsigned three word binary number using the RGDTB subroutine. These two binary fields are then multiplied together to form a three word product which is converted to a fourteen digit decimal number. This decimal field is then aligned to the decimal places specified in the result field and the field is stored at the assigned address.

Description of Operation

Initialization of working storage and saving of the index registers is performed on entry to the routine. The calling sequence of the caller is retrieved and stored for future information. Both factor 1 and factor 2 are then stored in the work areas in arithmetic format (without zones) and are converted to binary by RGDTB.

A loop to perform the multiplication of factor 1 by factor 2 in binary is then executed. Since the result field is a maximum of 14 digits, the product is formed in three binary words and then converted to a 14 digit decimal number through RGBTD. The resulting product has a number of decimal places equal to the sum of the decimal places of factor 1 and factor 2. This product is then aligned according to the number of decimal positions in the result field.

If half-adjust was specified, it is performed on the aligned result. This final number has zones placed with each of its digits so that it is in RPG object time data format and the field is stored in the assigned result field. The sign of the result is stored for the purpose of setting the resulting indicator as in ADD, and the index registers are restored to their original value. Control is then returned to the next sequential instruction in the calling routine.

External Routines Used

RGBTD, RGDTB

Work Areas:

MULT	Used to contain the word currently being used as multiplier.
SVWRK	Contains word of partial product during formation of the final result.
ODWRK	Contains word currently being used as the multiplicand.
OVFLW	Contains word of overflow in the Accumulator following each partial multiplication.
MLTPR	Contains three word binary multiplier.
MLTPC	Contains three word binary multiplicand.
WORK	Binary product is formed in this three word area.
FACT1	Contains the fourteen digit decimal factor 1.
FACT2	Contains the fourteen digit decimal factor 2.
Note:	The decimal result field is compiled within the area occupied by FACT1 and FACT2.

Calling Sequence

```
LIBF    RGMLT
DC      Address of Factor 1
DC      Address of Factor 2
DC      Address of Result Field
DC      Half-Adjust Indicator
```

<u>Divide (Chart MM)</u>

Description of Function

This subroutine has one entry point (RGDIV). The divide function causes the contents of the field in factor 1 (dividend) to be divided by the contents of the field in factor 2 (divisor). The result of this operation (quotient) is placed in the specified result field.

If factor 2 is zero, no processing is done and the error routine (RGERR) is called to give the operator the option of cancel or continue with the result field being set to zero. If factor 1 is zero, the quotient is zero and the remainder is zero.

Any remainder that results from this operation is lost unless the move remainder operation is specified as the next operation after divide. If a move remainder operation follows a divide operation, the result in the divide operation cannot be half-adjusted.

Description of Operation

When the subroutine is entered, initialization is performed to retrieve the calling sequence and the precision and scaling factors of factor 1, factor 2 and the result field. The input data (factor 1 and 2) is then converted from zoned decimal formal to decimal format without zones, and put into a 14 word field right justified with high-order zeros.

The decimal input fields are then converted to 6 word binary fields, one for the divisor (factor 2) and one for the dividend (factor 1). However, before the conversion to binary is performed, adjustment is performed if necessary.

The divisor and dividend are adjusted by adding zeros to the right of the units position of that field. The following formula determines whether or not field adjustment is necessary.

A = (Number decimal positions of result field) +
(Number decimal positions of Factor 2) -
(number decimal positions of Factor 1).

If A = 0, no adjustment takes place.
If A $>$ 0, the dividend will be adjusted.
If A $<$ 0, the divisor will be adjusted.

The amount of adjustment is determined by the absolute value of A in the preceding formula. The conversion to binary is accomplished by successive cumulative multiplications of each digit of the decimal number by A (base 16) from left to right.

Next the high-order three words of the divisor are checked; if non-zero, the divisor is assumed to be larger than the dividend. The dividend is then stored as the remainder and a return is made to the calling routine.

If the three high-order words of the divisor are zero, the three low-order words are checked next. At this time a pointer is adjusted to the high-order end of the temporary quotient (4 words long). Each time a word of zero is found in the divisor, the temporary quotient pointer is moved one word to the left. If the divisor is found to be completely zero, no processing is done, the result field is set to zero, and the subroutine RGERR is called. But, if a non-zero value was found, then the dividend is checked next for leading zero words. Each time a word of zeros is found, the temporary quotient pointer is moved one to the right. If the dividend is found to be zero, the quotient is zero and the remainder is zero, and a return is made to the calling routine.

At this time, the position of the temporary quotient pointer is tested. If the pointer is to the left of the 4 word field, the quotient is zero and the dividend is the remainder. If the pointer is to the right of the 4 word field, the quotient is zero and the remainder is zero. In both cases a return is made to the calling routine at this point.

If the pointer was within the range of the 4 word field the operation is continued by dividing the two high-order non-zero words of the dividend by the high-order non-zero word of the divisor. Load the result of the divide into the temporary quotient at the position pointed to by the temporary quotient pointer. (Temporary quotient is initially zero.)

Then the three low-order words of the divisor are multiplied by the four words of the temporary quotient. The product of the multiply is contained in a 7 word area. The 6 word dividend is then subtracted from the 7 word product of the last multiply. The result of this subtract is contained in a 6 word area.

This 6 word area is checked for leading zeros. For each leading zero, the temporary quotient pointer is moved one word to the right. When the first non-zero word is found, the position of the temporary quotient pointer is checked and if it is out of the temporary quotient area, the divide operation is complete. If not, a return is made to the divide routine to continue the loop.

There is only one variation to the loop: after the first store of the temporary quotient, the rest of the calculations made to the temporary quotient will be added to it or subtracted from it. This is determined by the subtract function. If the result of the subtract was negative, the alteration to the temporary quotient is added to it. If the result was positive, it will be subtracted.

Upon completion of the loop, the result of the last subtract becomes the remainder, and the temporary quotient is the final quotient. If half-adjust is specified, it will be performed at this time. Half-adjust is performed by comparing the remainder to the divisor. If the remainder is more than half as large as the divisor, the quotient will be increased by one. The quotient is then converted from a 3 word binary number to a 14 word decimal number, one decimal digit per word. The conversion is performed by dividing the binary number by A (base 16) and storing the remainder as a decimal digit. The decimal number is formed from right to left.

The decimal quotient is then moved to the result field specified by the calling sequence. A zone of 'F' will be attached to all digits unless the result is negative, in which case a sign of 'D' will be inserted. The sign of the result is determined by this rule: if the signs of factor 1 and factor 2 are equal the result is positive, if the signs of factor 1 and factor 2 are not equal, the result is negative.

The final step before returning to the calling routine is to store an indicator, as in ADD, which indicates the sign of the result.

External Routines Used

RGERR

Work Areas Used

- DIVDN: A 6 word area used for the binary dividend. (Factor 1)
- DIVSR: A 6 word area used for the binary divisor. (Factor 2)
- QUOTN: A 5 word area used for the temporary quotient in binary.
- FACT1: A 15 word area used by factor 1, factor 2 and the result field in decimal form.

The following are used by the multiply routine.

- MULT: Used to contain the word currently being used as the multiplier.
- SVWRK: Contains word of partial product for formation of final result.
- ODWRK: Contains the word currently being used as multiplicand.
- OVFLW: Contains word of overflow in the accumulator following each partial multiplication.
- WORK: Contains the 6 word result of the multiplication.

The following is used by the subtraction routine.

- REDIV: Contains the 6 word result of the subtract operation, and also contains the remainder at the end of the divide operation.

Calling Sequence

```
LIBF   RGDIV
DC     Address of Factor 1
DC     Address of Factor 2
DC     Address of Result Field
DC     Half-Adjust Indicator
```

Move Remainder (Chart MN)

Description of Function

The move remainder subroutine has one entry point (RGMVR) and is used to move the remainder from a divide operation to a separate field. If move remainder is used, it must immediately follow the divide operation. The result of a move remainder operation cannot be half adjusted.

Description of Operation

When the subroutine is entered, initialization is performed to retrieve the calling sequence and setup linkage with the divide routine. The remainder is retrieved for the divide routine in binary format, and converted (using RGBTD) to decimal. The decimal remainder is then adjusted to the specifications of the result field precision and scale. At completion of the adjustment the result is moved to the result field specified by the calling sequence. A zone of 'F' will be attached to all digits unless the result is negative, in which case a sign of 'D' will be inserted. The sign of the result of the move remainder operation is the same as the result of the divide operation.

External Routine Used

RGBTD

Work Areas Used: The move remainder routine has no work area of its own. A work area called Factl in the divide routine is used to hold the decimal remainder.

Calling Sequence:

```
LIBF   RGMVR
DC     Address of Result Field
```

RPG Conversion (Chart MO)

Description of Function

This subroutine has two entry points, RGBTD and RGDTB. RGDTB (decimal to binary) converts a 14 digit decimal number in arithmetic format (without zones or sign) to a positive 3 word binary number. This conversion is performed by successive cumulative multiplication of each digit of the decimal number by A (base 16) from left to right. RGBTD (binary to decimal) converts a 3 word unsigned binary number to a 14 digit decimal number. The conversion is performed by dividing the binary number by A (base 16) and storing the remainder as a decimal digit. The decimal number is formed from right to left.

Description of Operation

A common initialization routine is used for both RGDTB and RGBTD. This routine retrieves the parameter list and zeros the area in which the binary number is held. Entry point RGDTB retrieves a decimal number in arithmetic format and stores it in the save area. The binary portion already formed is multiplied by A (base 16) and the save area is added to the low order binary word. This procedure is continued until all the decimal digits have been converted.

Entry point RGBTD retrieves the unsigned three word binary number and divides it by ten. The remainder of the division is stored in the next low order position of the 14 word area in which the decimal number is being formed. This process is continued until the complete 14 word area has been filled with decimal digits.

Work Areas:

OVFW — The area which contains the portion of the product contained in the accumulator following the multiplication. It is added to the low order word of the next product.

WORK — The area used to hold the binary value of the number.

Calling Sequences:

```
LIBF   RGDTB
DC     Address of decimal number
DC     Address of binary number
LIBF   RGBTD
DC     Address of decimal number
DC     Address of binary number
```

Sterling Input Conversion (Chart MP)

Description of Function

The sterling input conversion subroutine has a single entry point (RGSTI). This subroutine takes a sterling field from the I/O area in a format consisting of pounds, shilling, pence, and decimal pence fields and converts it to pence and binary pence. The converted field is then stored in the area assigned to the field by the compiler in the RPG object time data format.

This routine properly converts IBM pence format and BSI pence and shilling formats to decimal form. It also supports both standard and non-standard sign positions. The zone of hexadecimal D is regarded as a negative zone while any other zone is regarded as positive. The zone representing the sign is undisturbed by the subroutine and is placed, in the format given in the data record, into the assigned field.

Description of Operation

The subroutine first retrieves the calling sequence and sets work areas within the program to correspond with the entries in the calling sequence. This is also done with the length and number of decimal places that are taken from the indicator word preceding the assigned field. From the specifications the number of pounds positions in the unconverted field is calculated.

The sign of the field is then retrieved by calculating the location of the word holding the sign in core storage. If a non-standard sign location is specified the column indicated in the calling sequence is used. If the sign is at the standard location, the number of pounds positions is added to the beginning field locations. If there are decimal pence places, the length to the low order decimal pence position is then added to this previous total. The column of the field in which the sign is located is now available and the routine at STAND determines the actual core storage location. The sign is then retrieved and stored.

The work area, BPENC, BINNO, and WORK are then initialized to binary zero and the field is placed into WORK. The pounds positions of the field are then put into arithmetic format by removing the zones in the field and then are converted to binary and stored at BINNO.

The value of the shilling field is determined in binary and then is multiplied by twelve to convert it into pence. This value is added to the value of the pence field in binary and is stored at BINNO. The pounds field is converted to binary by multiplying by 240. The total non-decimal pence is then formed and converted back to decimal and stored in WORK in arithmetic format. The zones and sign are inserted in the field and the field is stored in the assigned field position.

External Routines Used: RGMV1, RGBTD, RGDTB

Work Areas:

SIND	Set to 0 if IBM shilling format is used. Set to 1 if BSI shilling format is used.
PIND	Set to 0 if IBM pence format is used. Set to 1 if BSI pence format is used.
BPENC	Used to hold binary pence value of shilling and pence fields.
BINNO	Used to hold the binary value of the field.
WORK	Contains the decimal field both before and after conversion.

Calling Sequence

```
LIBF    RGSTI
DC      /xxxz where xxx = Column 1 of field
        in I/O area
        z = aabc where a = 0,
            (0 for IBM )         (0 for IBM)
        b = (shilling, ) and c = (pence     )
            (1 for BSI )         (1 for BSI )
            (shilling, )         (pence     )
DC      Address of assigned field
DC      /00E2 is standard, column-1 of sign
        position if non-standard.
```

Sterling Output Conversion (Chart MP)

Description of Function

The sterling output conversion subroutine has a single entry point (RGSTO). This subroutine takes a decimal field from an assigned field area and converts it to a format containing pounds, shillings, pence, and decimal pence fields. The converted field is then stored in the I/O area using subroutine RGMV2. This routine will convert to the format required for a printer file, for a field using IBM or BSI shilling format, or for a field using IBM or BSI pence format. If the field is to be edited, the output of the routine is stored back in the field location and the EDIT routine is called. This routine supports both the standard and non-standard sterling sign locations.

Description of Operation

The calling sequence is retrieved and the entries are placed in work areas within the routine. The scale and precision are taken from the word preceding the field to be converted. The address of the I/O area is taken from the RPG overhead area.

The field is moved from the assigned field location and placed in arithmetic format (without zones) in the work area WORK. The non-decimal pence positions are then converted to binary and stored in BINNO. This field is then divided by 12 and the remainder is converted into the specified pence format (print, IBM, or BSI). The converted pence field is placed in the work area WORK contiguous with the decimal pence.

The quotient of the above division is then divided by 20 and the remainder is converted into the proper shilling format. This field is then put into WORK. Both the pence and shilling fields for printed files have the tens position zero suppressed if zero suppress is not specified.

The quotient of the last division is then converted to decimal and stored in WORK. The converted field is moved from WORK to the assigned field, if EDIT must be used, inserting zones for RPG object time data format, and the routine returns to the caller. If EDIT is not used, zones are inserted in the pounds position and the decimal positions and zero suppression, if specified, is performed. The external routine RGMV2 is called to place the field in the I/O area. For a non-zero suppressed field, the location of the sign is calculated and the sign is inserted. Return to the calling sequence is then performed.

External Routines Used

RGMV2, RGBTD, RGDTB

Work Areas

SIND	Set to 0 if IBM shilling format is used. Set to 1 if BSI shilling format is used.
DIND	Set to 0 if IBM pence format is used. Set to -1 if BSI pence format is used. Set to +1 for printer fields. This field will be modified by +1 during execution of the program.
ZPIND	Set to 0 if zero suppress is used.
EDIND	Set to 0 if edit is used.
LPCNR	Used to control looping.
BINNO	Contains the field in binary.
WORK	Used to hold both the converted and unconverted fields in decimal.

Calling Sequence

```
LIBF    RGSTO
DC      /xxxz where xxx = column-1 of field
        in I/O area
        z = aabc where a = 0
            (0 for IBM,  )          (0 for IBM )
            (printed     )          (printed   )
        b = (shilling    ) and c =  (pence     )
            (1 for BSI   )          (1 for BSI )
            (shilling    )          (pence     )
DC      Address of assigned field
DC      /xxyy where xx = column-1 of sign
        position if non-standard and /E2
        if standard.
             ( 00 - Field not to be printed
             ( F0 - Field to be printed but
             (        not edited
        yy = ( F5 - Field to be printed with
             (        zero-suppress
             ( FF - Field to be printed with
             (        edit
```

Edit (Chart QX)

Description of Function

This subroutine has one entry point (RGEDT). The edit routine takes a decimal field held in RPG object time format and a specified edit word and performs editing on the decimal field. The resulting edited field is then stored in the specified location in the I/O area. The routine performs either zero-suppression or asterisk protection. Either fixed or floating dollar sign may be used.

All characters other than the digit select characters, the significant start character, and an ampersand (which causes a blank) will be printed if to the right of the first significant digit, and will be replaced by a blank if to the left. The letters CR or a minus sign in the status portion of the edit word will be undisturbed if the field is negative and will be replaced by blanks if positive.

Description of Operation

The calling sequence is retrieved and the information therein is stored. The address of the I/O area is taken from the RPG overhead area and the length of the field is taken from the indicator word preceding the assigned field. If a fixed dollar sign is specified, a dollar sign is placed as the first character in the edited area. A digit from the field is then retrieved. If the digit is non-zero, SWITC is turned on. If the digit is zero and SWITC is off, the fill character replaces the digit. The digit is stored and a character from the edit word is retrieved.

If the character is a digit select character it is replaced in the edited result by the digit or fill character. If it is a significant start character, SWITC is turned on and the digit is selected. If it is any other character, it is placed in the edited field if SWITC is on. If SWITCH if off, it is replaced in the edited field by the fill character. This process is continued until all the characters in the edit word have been used.

The sign of the field is then interrogated. If it is positive, CR or - is blanked out. The edited result is then placed in the I/O area by routine RGMV2.

If floating dollar sign was specified it is placed in the character immediately to the left (of either the significant start character or) of the first significant digit.

External Routines Used

RGMV2

Work Areas:

SWITC Used to indicate whether or not a significant digit or significant start character has been encountered.
RESLT Used to hold the edited field.

Calling Sequence

```
LIBF    RGEDT
DC      /XXXZ where XXX = column-1 of field
        in I/O area
        Z = yayb where y = 0, a = 1 if CR,
        0 if minus; b = 1 if sterling,
        0 if decimal
DC      Address of assigned field
DC      Address of edit word
DC      /XXYY where XX - length of edit
        word, YY = number of replacement
        characters
DC      /XXYY where XX = column-1 of CR/MIN
        in edit word,
             ( 00 if no dollar sign
        YY = { F0 if fixed dollar sign
             ( FF if floating dollar sign
```

Sequential Access (Chart MR)

Description of Function

The sequential access subroutine has three entry points:

SEQOP Entry point for the open function
SEQIO Entry point for the I/O function, and
SEQCL Entry point for the close function, and is used to sequentially input/output/update records of a sequentially organized disk file.

Description of Operation

SEQOP: The open entry point begins by performing a check on the buffer address to ensure that it is on an even word boundary and issues a diagnostic if the address is incorrect. Next, the record size and number of records per sector are checked to ensure their accuracy; they are diagnosed if incorrect.

If the DFI indicator has an output file indicator code, the routine sets the record address pointer to the buffer location for the placement of the first record, sets the record number to one, and blanks out the data buffer. Processing continues for all file types by setting the return code to indicate the file is open and returning to the next object program instruction.

SEQIO: The I/O function entry point begins by testing the return code in the DFI table to determine if the file has been opened. If the file is not open, a check is made to determine if a terminating error had been issued and if one had, a diagnostic is issued. Otherwise, processing continues as though the file was successfully opened without a previous error. The first time the routine is entered a check is performed to ensure the read/write indicator is a read code and a diagnostic is issued if it is not a read code.

If the file is an update file and a read is indicated by the read/write indicator, the routine follows the path of an input file. However, if a write is indicated for the update file, the routine sets a switch to indicate a record has been updated in the present buffer. This switch is later tested when the buffer has been processed and a read is indicated. If the switch is on, the routine performs a disk write before the disk read.

If the write indication for the update file is on when the buffer is completely processed, then the disk write is performed by the routine immediately. When the file is an input file, the routine reads a sector of data into the buffer and sets a pointer and record counter to the first record in the buffer. The pointer and record counter are updated during successive reads until the buffer is processed, at which point another sector of data is read.

End of file is tested on each read and an indicator returned when encountered. If the file is an output file, the routine initially sets the pointer to the buffer location where the first record is to be placed and increments the pointer by the record size for each successive write until the buffer is full, at which time the buffer is written on disk. The pointer is reset to the beginning of the buffer and the data buffer is blanked out. The routine returns to the next object program instruction after retrieval of each record.

SEQCL: The close entry point is entered when processing for the file is complete. The routine begins by determining file type. If the file is an input type, it is closed by entering a code (.I) in the return code word of the DFI table and returns to the calling sequence.

If the file is an output type, the EOF record is suffixed to the data and written on the file, and the file is closed by entering a code (.O) in the return code word of the DFI table and a return is made to the calling sequence.

If the file is an update file, the update-write switch is checked to determine if a disk-write for the data buffer is necessary. If it is, the write is performed. Then the file is closed by entering a code (.U) in the return code word of the DFI table and a return is made to the calling sequence.

Calling Sequences

```
LIBF    SEQOP
DC      Address of the DFI table
LIBF    SEQIO
DC      Address of the DFI table
LIBF    SEQCL
DC      Address of the DFI table
```

Direct Access (Chart MS)

Description of Function

The direct access subroutine has three entry points:

- DAOPN Entry point for the open function,
- DAIO Entry point for the random retrieve/update function,
- DACLS Entry point for the close function,

and is used to randomly retrieve and/or update records on a sequentially organized disk file.

Word	Contents
0,1,2	DSA
3	The number of records per sector. The maximum entry is /0140 (320 one-word records).
4	The length of each record in words. The maximum entry is /0140 (one 320-word record).
5	/0000 for a read operation, /0001 for a write operation.
6	The address of the data buffer. This address must be on an even-word boundary.
7	.I for an input function, .O for an output function, .U for an update function.
8	The record number of the record being processed.
9	The return code * for this operation.
10	The address of the record being processed.

*Return codes for sequential access are as follows:

Return Code	Meaning
/5555	File open
/8010	Disk file is full
/8011	Write indicator with input file
/8012	Read indicator with output file
/8013	Record size exceeds sector size
/8014	Number of records per sector not maximum
/8015	File accessed when not open
/8016	Buffer not on even-word boundary
/8017	Write before read
/FFFF	End of File
/0FFF	File closed

Table 13. DFI Table for the Sequential Access Subroutine

Description of Operation

DAOPN: The open entry point begins by checking the buffer address to ensure it is on an even-word boundary. A diagnostic is issued if the address is incorrect. Next the record size and number of records per sector are checked to ensure they are correct. Again, diagnostics are issued if the items are incorrect. The return code is set to indicate the file is opened and the routine returns to the calling sequence.

DAIO: To randomly retrieve/update records, the routine at this entry point begins by checking the DFI table to ensure that the file has been opened. A diagnostic is issued if the file is not opened or if the return code contains a terminating error type code. The relative record number is checked to determine if it is negative or if it is outside the bounds of the file and diagnosed if either error is found.

A calculation is performed with the relative record number to determine the sector address of the record. If the determined sector address is the same as the one previously read and the operation is a write, the routine performs a disk write and returns to the calling sequence. If the determined sector address is the same as the one previously read and the operation is a read, the routine updates the record address pointer and returns to the mainline. If the determined sector address is not the same as the one previously read, the operation must be a read (a diagnostic is issued if a write is indicated). The routine then performs a disk read, updates the record address pointer and returns to the calling sequence.

DACLS: This entry point is entered after the records have been retrieved/updated. It sets the return code in the DFI table to indicate the file is closed and returns to the calling sequence.

Calling Sequences

```
LIFB    DAOPN
DC      Address of the DFI Table
LIBF    DAIO
DC      Address of the DFI Table
LIBF    DACLS
DC      Address of the DFI Table
```

Word	Contents
0,1,2	DSA
3	The number of records per sector. This entry must be the same as the number of records per sector on the file being accessed.
4	The length of the record in words. This entry must be the same as the length of the records on the file being accessed.
5	/0000 for a read operation, /0001 for a write operation.
6	The address of the data buffer. This address must be on an even-word boundary.
7 & 8	The record number of the record being processed. Position 6 will be all zeros for record numbers of less than 65,536.
9	The return code * for this operation.
10	The address of the record being processed.

*Return codes for direct access are as follows:

Return Code	Meaning
/5555	File open
/8000	Record number not in file
/8001	Record size not within limits
/8002	Records per sector not maximum
/8003	Record number not positive
/8004	Write before read
/8005	File accessed when not open
/8006	Buffer not on even-word boundary
/0FFF	File closed

Table 14. DFI Table for the Direct Access Subroutine

ISAM LOAD (CHART MT)

Description of Function

The ISAM LOAD subroutine has three entry points:

ISLDO	Entry point for the open function,
ISLD	Entry point for the loading function,
ISLDC	Entry point for the close function,

and is used to load records onto an indexed sequentially organized disk file.

Description of Operation

ISLDO: The open function is the first to be executed when loading an ISAM file. Execution of the open function begins by checking the type code (/1111) in the DFI table to ensure it is a load file. A diagnostic is issued if the type code is incorrect. The record size, number of rerecords per sector, and key length are checked next to guarantee they are within the accepted limits. Diagnostics are issued if any of these are incorrect.

Next, the index size, number of index entries per sector, index buffer address and data buffer address are checked. The buffer addresses must be on even word boundaries, and the index size and number must be within the accepted limit or a diagnostic is issued. The diagnostics are codes returned in the return code word of the DFI table.

If all of the above tests are satisfied, the routine continues by setting up the index entry pointer, beginning index sector address, beginning Prime Data sector address, Prime Data record pointer, zeroing out the key hold area, and blanking out the data buffer. (The sector addresses in the index and label are the relative sector addresses from the beginning of the file.)

The routine then sets the return code word in the DFI table to indicate the file is open and returns to the calling sequence.

ISLD: This entry point is used each time a record is to be loaded onto the ISAM file. When entered, the routine begins by checking the return code in the DFI table to ensure the file is opened and that there are no terminal type errors in the return code. The key of the present record is then checked against the key in the hold area to ensure the records are in sequence. The control field of the last record is set to binary zeros.

Then a test is performed to determine if the P.D. buffer is full and if it is not full, the routine adds 1 to a record counter, increments the P.D. record address pointer to the location where the next record is to be located, and returns to the calling sequence. If the buffer is full, the routine performs another test which determines if the sector address of the next disk write is an end of cylinder sector or not. If it is not the end of a cylinder, the routine writes the P.D. buffer on disk, adds 1 to the sector address, sets the P.D. record address pointer to the beginning of the buffer, sets the record counter back to one, and returns to the user.

If the sector address was an end of cylinder sector, the routine moves the key from the last record in the buffer to the key portions of the next index entry. A check is made on the index buffer to determine if it is full and if it is, writes the buffer on disk, and resets the index pointer and counter. Otherwise, the index pointer and counter are incremented, the data buffer is written on disk, the data buffer is then blanked out, the P.D. pointer and counter is reset, the sector address is incremented by one and the routine returns to the calling sequence.

ISLDC: This entry point is entered when all records have been loaded and the ISAM is to be closed. After all the records have been loaded, the ISAM file must be closed by entering the Load Routine at the ISLDC entry point. The close function generates the end of file record (all 1 bits), builds the last index entry (the key portions of which are all 1 bits), writes the last sector of index and data buffer, builds the ISAM file label in the index buffer, writes the label on the first sector of the file, sets the return code word in the DFI table to indicate the file is closed and returns to the calling sequence.

Calling Sequences

```
LIBF   ISLDO
DC     Address of the DFI table
LIBF   ISLD
DC     Address of the DFI table
LIBF   ISLDC
DC     Address of the DFI table
```

Word	Contents
0,1,2	DSA
3	The key length in characters. Maximum is /0032.
4	The length of the record in words. The maximum entry is /0140 (one 320-word record).
5	The address of the index buffer. This address must be on an even-word boundary.
6	The address of the data buffer. This address must be on an even-word boundary.
7	/1111 to identify the ISAM load routine.
8	The number of sectors required for the index.
9	The return code * for this operation.
10	The address of the record being processed.
11	The address of the index entry.
12	The number of index entries per sector. This value must be the maximum number of index entries that will fit on a sector.
13	The index entry length in words.
14	The number of records per sector. This value must indicate the maximum number of records that will fit on a sector.
15	The prime data record number.
16	The index entry number.
17	The address of the key hold area. After the close routine has been executed this location contains the sector address of the last prime data sector.
18	The sector address of the last index sector.
19	The sector address of the next overflow sector.
20	The record number of the next overflow record.

Table 15. DFI Table for the ISAM LOAD Subroutine

*Return codes for ISAM LOAD are as follows:

Return Code	Meaning
/5555	File is open
/8020	Not a load function
/8021	Record size or number of records per sector incorrect
/8022	Key length greater than maximum
/8023	Index entry length not same as length computed from key length
/8024	Number of index entries per sector incorrect
/8025	Prime data area is full
/8026	Index area is full
/8027	File is not open
/8028	Index buffer not on even-word boundary
/8029	Data buffer not on even-word boundary
/802A	Input record out of sequence
/0FFF	File is closed

ISAM ADD (CHART MU)

Description of Function

The ISAM ADD subroutine has three entry points:

ISADO Entry point for the open function,
ISAD Entry point for the ADD function,
ISADC Entry point for the close function,

and is used to add records to an indexed-sequentially organized disk file.

Description of Operation

ISADO: This entry point begins by checking tye type code in the DFI table to ensure the file is an ADD type (/000). A diagnostic is issued if the type code is incorrect. The index buffer address is checked to ensure it is on an even word boundary and diagnosted if it is incorrect.

Next the ISAM file label is read into the index buffer. The key length and record size from the label is compared with the key length and record size in the DFI table; if they are not equal a diagnostic is issued. Otherwise, the information contained in the ISAM file label is moved to the DFI table.

The first sector of index is then read into the index buffers, the index entry pointer is set, the index entry number is set to 1, the add record area is blanked out and the return code is set to indicate the file is opened. The routine then returns to the calling sequence.

ISAD: This entry point is entered each time a new record is to be added to the ISAM file. Upon entering the routine, the return code in the DFI table is checked to determine if the file is open, or if there are any terminal type errors in the return code. If the file is not open or if there are terminal errors, a diagnostic is issued. Next the index entry to govern the new record is located, and from that entry it is determined where the new record is to be placed. This new record can be placed in either of the following:

1. P.D. area, or
2. Overflow area.

When the new record is to be placed in the Prime Data area, the routine searches the cylinder, sector by sector, until the sequential location of the record is found. The last record on this cylinder is then moved to the next available overflow sector and the index is updated.

The records are then shifted one record location at a time to the right until the location of the add record is vacated. The add record is then moved into this location and return is made to the calling sequence. The addition of add records to the Prime Data area is done in this fashion so that records will not be lost in case of a CPU or disk error during an add function. Duplicate records can occur with this method.

When the new record is to be placed on the overflow area, the routine searches through the chain of overflow records until the sequential location of the new record is found. The new record is placed on the next available overflow sector and either the index is updated or the sequence-link control fields of the records are updated to keep the chain of overflow records in sequence. The sector addresses in the index or sequence-link control field are relative sector addresses from the beginning of the file. The routine then returns to the calling sequence.

Before each non-error return, the add record area is blanked out.

ISADC: This routine builds an updated ISAM file label in the index buffer and writes it on the first sector of the file. The routine sets the return code word in the DFI table to indicate the file is closed and returns to the calling sequence.

Calling Sequences

```
LIBF   ISADO
DC     Address of the DFI table
LIBF   ISAD
DC     Address of the DFI table
LIBF   ISADC
DC     Address of the DFI table
```

Word	Contents
0,1,2	DSA
3	The key length in characters. Maximum length is /0032.
4	The length of the record in words. The maximum entry is /0140 (one 320-word record).
5	The address of the index buffer. This address must be on an even-word boundary.
6	The address of the record being added to the file.
7	/0000 to identify the ISAM add routine.
8	The index entry number in process.
9	The return code * for this operation.
10	The prime data record number in process.
11	The address of the index entry.
12	The number of index entries per sector.
13	The index entry length in words.
14	The number of records per sector
15	The record number of the last prime data record processed.
16	The number of the last index entry for the file.
17	The sector address of the last prime data sector.
18	The sector address of the last index sector.
19	The sector address of the next overflow sector.
20	The record number of the next overflow record.

*Return codes for ISAM ADD are as follows:

Return Code	Meaning
/5555	File is open
/8030	Not an add function
/8031	File is not open
/8032	Key length in DFI table not same as key length in label
/8033	Record length in DFI table not same as record length in label
/8034	Key is presently on file
/8035	Overflow area is full
/8036	Index buffer not on even-word boundary
/0FFF	File is closed

Table 16. DFI Table for the ISAM ADD Subroutine

ISAM Sequential (Chart MV)

Description of Function

The ISAM Sequential subroutine has four entry points:

ISEQO	Entry point for the open function,
ISETL	Entry point for the set lower limit function,
ISEQ	Entry point for the retrieve/update function,
ISEQC	Entry point for the close function,

and is used to sequentially retrieve/update records on an ISAM file.

Description of Operation

ISEQO: The open entry point begins by checking the type code word in the DFI table to ensure it is a sequential processing type file. A diagnostic is issued if the type code is incorrect. Next the index buffer address and data buffer address are checked to ensure they are on even-word boundaries and a diagnostic is issued if they are incorrect.

The ISAM file label is read into the index buffer next. The key length and record size from the label are compared with the key length and record length from the DFI table to ensure they are equal. Again, a diagnostic is issued if either is incorrect. The needed information is moved from the label to the DFI table, and the first sector of index is read into the index buffer.

The index address pointer is set to the first entry and the index entry number is set to one. The prime data number is then set to zero, the return code is set to 5555 to indicate that the file is open and the routine returns to the calling sequence.

ISETL: ISEQ is entered next if the file is to be processed from the beginning. If the processing is to begin at a lower limit, the ISETL entry point is entered. In this routine, after a check to see if the file is open, a search of the index is performed until the entry governing the low limit key is found. A test is performed to determine if the low limit record is located on the prime data or the overflow area. The determined area is searched until the record or the next higher record (if low limit is not on file) is found, and if the record is in the overflow area, a switch is set to indicate the record is in that area. The SETL switch is set on, the PD record number and pointer are set to the found record, the index pointer and number are set, the update-write switch is set off, and the routine returns to the calling sequence.

ISEQ: This entry point is entered to sequentially retrieve/update records on an ISAM file. This routine starts by making sure that there are no terminal type errors in the return code. If the SETL switch is on, it is turned off and the routine returns to the user because the record is presently available from the ISETL routine.

Next, an update with a write is checked and if this condition exists, the record is written and the return is to the calling sequence. Next, the record number is checked; if zero, which indicates the first time, the first prime data sector is read and the record address and number are set. If the record number is not zero, the overflow switch is checked and if it is off, the next record is retrieved from the prime data area except when end of cylinder, in which case the record is retrieved from the overflow chain or from the next prime data cylinder.

If the overflow switch is on, the next record is retrieved from the overflow chain. If the present record is the end of overflow chain, the next index entry is retrieved, the prime data sector is read in, and the record address and number are set. After any record is retrieved, end of file is checked and if found, the return code is set to hex FFFF.

ISEQC: The routine at this entry point causes the file to be closed by writing a sector if an update has been requested but no write has been performed yet. The return code is then set to 0FFF to indicate the file is closed and a return is made to the calling sequence.

Calling Sequences

```
LIBF   ISEQO
DC     Address of the DFI table
LIBF   ISETL
DC     Address of the DFI table
LIBF   ISEQ
DC     Address of the DFI table
LIBF   ISEQC
DC     Address of the DFI table
```

Word	Contents
0,1,2	DSA
3	The key length in characters. Maximum length is /0032.
4	The length of the record in words. The maximum entry is /0140 (one 320-word record). The record length must be the same as the record length in the file label.
5	The address of the index buffer. This address must be on an even-word boundary.
6	The address of the data buffer. This address must be on an even-word boundary.
7	/0001 to identify the ISAM sequential retrieve routine. /0010 to identify the ISAM sequential update routine.
8	The address of the key hold area if processing starts at a point other than the first record in the file. If the entire file is being processed, the value is /0000.
9	The return code * for this operation.
10	The address of the record in process.
11	The address of the index entry used to locate the record.
12	The number of index entries per sector.
13	The index entry length in words.
14	The number of records per sector.
15	The update-write indicator.
16	The number of the index entry in process.
17	ISETL switch to indicate low limit record found.
18	The internal switch used to indicate that the last record in the overflow area has been found.
19	The read/write indicator. If the routine type (position 6) is retrieve, this entry should be /0000. If routine type is update, this entry is /0000 for retrieve and /0001 for update.
20	The prime data record number in process.

*Return codes for ISAM sequential are as follows:

Return Code	Meaning
/5555	File is open
/8040	Not a sequential retrieve or update function
/8041	Index buffer not on even-word boundary
/8042	Data buffer not on even-word boundary
/8043	Key length in DFI table not same as key length in label
/8044	Record length in DFI table not same as record length in label
/8045	File not open
/8046	Write before read on update
/FFFF	End of file
/0FFF	File is closed

Table 17. DFI Table for the ISAM Sequential Subroutine

ISAM Random (Chart MW)

Description of Function

The ISAM random subroutine has three entry points:

ISRDO Entry point for the open function,
ISRD Entry point for the retrieve/update function,
ISRDC Entry point for the close function,

and is used to randomly retrieve/update an ISAM file.

Description of Operation

ISRDO: The routine at this entry point begins by checking the type code to ensure the file is random retrieve/update. If incorrect, a diagnostic is issued.

Next the addresses of the two buffers are checked to ensure they are on even-word boundaries. If either is incorrect, a diagnostic is issued. The ISAM file label is read into the index buffer and the key length and the record length are checked against the entries in the DFI. If they are not the same, diagnostics are passed back to the user. Information from the label is then moved to the DFI and the correct word counts are placed in the index and data buffers.

The first index sector is read and the pointers are set to the first index entry. The first-time switch is set on, hex 5555 is placed in the return code to indicate the file is open, and the routine returns to the calling sequence.

ISRD: The routine at this entry point starts by checking the return code for any error that would hinder processing. If none is found, a check is made for an update file with a write. If found, the write is performed after which the routine returns to the calling sequence.

If the condition is not update with write, a check is made of the first-time switch. If this switch is on, a search is made of the index from the beginning. If the switch is off, the requested record key is checked against the present index entry.

If the key is less than the present index, the search begins again from the first index entry. If the key is equal to or greater than the present index, the search continues from this point. When the correct index entry is found, the key of the requested record is compared against the prime data key in the index entry to determine if the record is on the prime data area or in the overflow chain.

When this has been determined, a search is made of the area. When the record is found, the address of the record is set and the routine returns to the calling sequence.

If the record is not found, the return code is set to indicate no record found and the routine returns to the calling sequence.

ISRDC: The routine at this entry point sets the return code to indicate that the file is closed and returns to the calling sequence.

Calling Sequences

```
LIBF    ISRDO
DC      Address of the DFI table
LIBF    ISRD
DC      Address of the DFI table
LIBF    ISRDC
DC      Address of the DFI table
```

Word	Contents
0,1,2	DSA
3	The key length in characters. The maximum length is /0032. The key length must be the same as the key length in the file being accessed.
4	The length of the record in words. The maximum entry is /0140 (one 320-word record). The record length must be the same as the record length of the file being accessed.
5	The address of the index buffer. This address must be on an even-word boundary.
6	The address of the data buffer. This address must be on an even-word boundary.
7	/0100 identifies the ISAM random retrieve routine. /1000 identifies the ISAM random update routine.
8	The address of the key hold area containing the key of the record to be processed.
9	The return code * for this operation.
10	The address of the record in process.
11	The address of the index entry used to locate the record.
12	The number of index entries per sector.
13	The index entry length in words.
14	The number of records per sector.
15	The prime data record number.
16	The number of the index entry in process.
17	A first-time switch. This switch is set off after one record has been processed.
18	An internal switch used to indicate the record found is in the overflow area.
19	A read/write indicator. If the routine type (position 6) is retrieve, this entry should be /0000. If routine type is update, this entry is /0000 for retrieve and /0001 for update.

*Return codes for ISAM random are as follows:

Return Code	Meaning
/5555	File is open
/8050	Not a random retrieve or update function
/8051	Index buffer not on even-word boundary
/8052	Data buffer not on even-word boundary
/8053	Key length in DFI table not same as key length in label
/8054	Record length in DFI table not same as record length in label
/8055	File is not open
/8056	Write before read on update
/8057	Record not on file
/0FFF	File is closed

Table 18. DFI Table for the ISAM Random Subroutine

CORE DUMP TRACE OF AN OBJECT PROGRAM

This section is presented as an aid to those persons who have occasion to examine areas of a core storage dump of an RPG object program.

Figure 24 shows the source statements for a disk to printer program and four maps which are produced by the compiler. These maps show the hexadecimal address of indicators, field, literals, and key routines relative to the beginning of the generated mainline program. These maps are identified by ①, ②, ③, and ④ in Figure 24.

The Core Load Builder, if requested, will produce a core map showing the actual core locations for the core load, the system subroutines address and the mainline program execution address. These addresses are identified by ⑤ and ⑥ in Figure 24.

To find the beginning of the RPG generated mainline program, subtract hexadecimal 11 from the execution address. The actual core load addresses for the RPG indicators, fields, literals and key routines are calculated as follows:

- Subtract hexadecimal 11 from the execution address and then add the address of the desired indicator, field, literal, or key routine.

For example, the actual address of the Total Lines routine (Figure 25) is calculated:

```
•  020F   (execution address)
  - 111
   01FE
   047D   (total lines address)
   057B   (actual core address)
```

The RPG Core Image Header is located at the execution address minus hexadecimal 002F. (Further information on the Core Image Header is contained in IBM 1130 Disk Monitor System, Version 2, Programming and Operator's Guide, Form C26-3717.)

The pseudo register area is located at the execution address minus hexadecimal 0010.

The 28 word Key Routine address (FAT) table is located at the execution address plus hexadecimal 0002.

The fixed length routines (Fixed Driver) of the mainline program are located starting at the execution address plus hexadecimal 001E.

The variable length routines of the mainline program start at the execution address plus hexadecimal 013F.

To calculate the address of the current I/O area for a file, PRINT, defined in Figure 24 will be used as an example. Since this is the second file defined in the program, the IOD address for this file is given in the key addresses map as File Seq 2. (File Seq and IOD are interchangeable in meaning and usage.) The address of File Seq in this case is hexadecimal 0224.

To this address must be added the execution address minus hexadecimal 11:

```
•    0224   (address of File Seq 2)
   + 020F   (execution address)
     0433
   -   11
     0422   (address of the PRINT IOD)
```

Finally, the address of the current I/O area is found in word4 of the IOD, so a 3 is added to the IOD address and location 0425 contains the address of the current I/O area for the PRINT file (03E6).

The address of the DFI table for a disk file may be found by obtaining the IOD address for the file and adding hexadecimal 6 to that address. Again referring to Figures 24 and 25 INDISK is the first file defined in the program and its address (03A4) is listed for File Seq 1 under Key Addresses. To this address is added the execution address minus hexadecimal 11 to obtain the IOD address for this disk file. To the IOD address a 6 is added to obtain the location that contains the address of the DFI Table. The following steps summarize this calculation:

```
•    03AE   (address of File Seq 1)
    +020F   (execution address)
     05BD
   -   11
     05AC   (address of IOD for INDISK)
   +    6
     05B2   (address of DFI table for
            INDISK FILE)
```

Further information on the format and content of DFI tables is contained under Library Subroutines.

```
PAGE    2
                                              V1-0  1130  RPG   RPGOBJ

  SEQ NO  PG LIN    SPECIFICATIONS COL 6 - 74                                                    ERRORS

                    H
   0001             FINDISK  IPE F      120           DISK
   0002             FPRINT   O   F      120           PRINTER
   0003             IINDISK  AA  01
   0004             I                                       1 120 RECORD
   0005             OPRINT   D  1      01
   0006             O                          RECORD   120
   0007             O                                    20 'RPG'


PAGE    3
                                          INDICATORS (1)

IND    DISP        IND    DISP        IND    DISP        IND    DISP        IND    DISP        IND    DISP

MR     0150        00     0151        OF     0152        OV     0153        1P     0154        L0     0155
L1     0156        L2     0157        L3     0158        L4     0159        L5     015A        L6     015B
L7     015C        L8     015D        L9     015E        LR     015F        H1     0160        H2     0161
H3     0162        H4     0163        H5     0164        H6     0165        H7     0166        H8     0167
H9     0168        01     0169

                                          FIELD NAMES (2)

FIELD     DISP  L  T D     FIELD     DISP  L  T D     FIELD     DISP  L  T D     FIELD     DISP  L  T D

RECORD  016A 120 A

                                               LITERALS (3)

          LITERAL                  LENGTH   TYPE   DISP          LITERAL              LENGTH   TYPE   DISP
RPG                                   3       A     01E3
                                          KEY ADDRESSES OF OBJECT PROGRAM (4)

   NAME OF ROUTINE                 HEX DISP                NAME OF ROUTINE                 HEX DISP

      H + D LINES                    046F                     TOTAL LINES                    047D
      DETAIL CALCS                   046F                     TOTAL CALCS                    047D
      CHAIN ROUT 1                   0410                     LOW FIELD                      0426
      EXCPT LINES                    048B                     CLOSE FILES                    058F
      FILE SEQ 1                     03AE                     FILE SEQ 2                     0224

END OF COMPILATION

// XEQ        L              R
R 41   11F0 (HEX) WDS UNUSED BY CORE LOAD (5)
CALL TRANSFER VECTOR
 RGERR   0C04
 EBPT3   09B4
LIBF TRANSFER VECTOR
 RGMV2   0CDE
 RGMV1   0C4A
 SEQIO   0AB0
 SEQCL   0B65
 SEQOP   0A34
 ZIPCO   0914
 PRNT1   0792
SYSTEM SUBROUTINES
 ILSX4   0D9B
 ILSX2   0DBD
 ILSX1   0DD6
      020F (HEX) IS THE EXECUTION ADDR. (6)
```

Figure 24. Object Program Core Dump Trace

01E0	020F	0000	FFFF	0000	001E	0000	01E0	0000	001D	0C2F	1F7E	0091	0DD6	0DBD	0091	0D9B	CORE IMAGE HEADER
01F0	0091	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	2000	0000	0000	72FF	0000	
0200	0000	03E6	01FE	0000	0000	0000	0000	0697	076C	022F	0773	0000	0000	06A8	0427	4C00	PSEUDO REGISTERS; EXECUTION ADDRESS
0210	0785	0785	0310	066D	0770	067B	0776	066D	0678	060E	01FE	01FE	01FE	01FE	01FE	01FE	
0220	01FE	01FE	0624	060E	0689	01FE	077F	078D	063B	6EFE	063B	259F	736E	7401	020A	7403	FAT
0230	020A	C480	020A	D0CC	4C80	0200	6580	020E	C00C	D101	C400	0329	4CA0	020E	C007	D107	
0240	4480	1FFE	1111	4C00	0247	0000	7009	C400	035D	4C20	02F2	6500	035E	C01C	D01C	C100	
0250	4C20	02C2	7101	74FF	026B	70F9	6500	0352	C013	D013	1010	D100	7101	74FF	026D	70FB	
0260	C0C1	D0A7	C480	0209	D0A2	C008	D400	0353	4C80	0207	0009	0000	0016	0000	0001	C400	
0270	035D	90FC	4C18	0301	6580	0222	6D00	0200	C102	D400	0207	C100	D400	0201	6580	0201	
0280	C106	D400	0201	6580	0201	C103	D400	0201	C400	034D	180F	D400	034E	1810	D400	034D	
0290	70D7	6580	0209	C103	D002	C0D8	D400	0367	C400	0202	9400	020E	4C18	0317	6580	0209	FIXED DRIVER (MAINLINE)
02A0	C101	D400	0207	70C4	6580	0207	C101	D400	020A	6D00	020D	C100	D400	020E	6580	020E	
02B0	C103	D400	0201	C101	D400	020E	C004	D400	0209	4C80	020E	022D	6580	020A	C101	D400	
02C0	0207	70A6	C01D	D400	020D	C015	D400	020E	7401	020D	7401	020E	C480	020D	9400	026E	
02D0	4C20	02C8	C00C	EC00	020E	D002	4480	1FFE	1120	4C00	0256	0000	0000	0000	0000	1120	
02E0	035D	C00F	4C18	02EB	6580	020E	C104	E009	9008	4C18	026F	C400	0218	D400	0207	4C80	
02F0	0207	0000	6500	0353	C100	D101	7101	74FF	0800	70FB	C400	0218	D400	0207	4C80	0207	
0300	000A	C400	0212	D400	0203	1010	D400	0205	D400	0206	6500	0310	6D00	0208	4C80	0203	
0310	4C80	0227	6038	0000	0000	0000	01FE	1000	70D2	C003	D0FC	4C00	026F	1000	6580	020E	
0320	C101	C007	4C00	0236	0000	00F1	01FE	01FE	0000	0000	D001	4400	0000	0000	6804	C003	
0330	804C	D001	4400	0000	C053	4804	0000	7014	C055	8042	D053	6804	C003	4C00	088D	0000	
0340	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0001	INDICATORS
0350	0000	0000	0000	0001	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
0360	0000	0000	0000	0000	0000	0000	0000	0001	77FF	0000	0040	0000	00C5	0000	0040	000E	FIELDS
0370	00FF	0000	00D9	0000	0000	0000	0040	0000	00C5	0000	0040	0000	00C7	0000	0040	0000	
0380	00C9	0000	0040	0000	00E2	0000	C000	0000	C040	0000	00E3	0000	0040	0000	00C5	0000	
0390	0040	0000	00D9	0007	00FF	0000	00C3	0000	0000	0000	00E4	0000	00E2	0000	00E3	0000	
03A0	00D6	0000	00D4	0000	00C5	0000	00D9	0015	00FF	0000	0000	0000	00D3	0000	00D6	0000	
03B0	00C3	0000	00C1	0000	00E3	0000	00C9	0000	00D6	0000	00D5	0000	0000	0000	0040	0000	
03C0	0040	0000	0040	0000	0040	0000	C040	0000	0040	0000	0040	0000	00C9	0000	0000	0000	
03D0	00D5	0000	00E5	0000	00D6	0000	00C9	0000	00C3	0000	00C5	0016	00FF	0000	00C9	0000	
03E0	0000	02FF	00D9	00D7	00C7	003C	C0F0	00F0	0000	00F0	00F0	00F0	00F0	17FF	00D9	D7C7	I/O AREA
03F0	0040	0000	0040	0040	0040	00C1	0040	00C3	0040	00C3	0000	0040	00D6	0040	00E4	0040	
0400	00D5	0040	00E3	0000	0040	00E2	0040	0040	00D9	17FF	0040	00C5	0000	0040	00C3	0040	
0410	00C5	0040	00C9	0040	00E5	0000	0040	00C1	0040	00C2	0040	00D3	0040	00C5	0000	0040	
0420	0040	00D9	0446	0000	0475	03E6	0427	C0[illegible]	[illegible]80	0445	D01A	6918	6931	7101	69F6	6919	FILE SEQ2 IOD; Variable Section (Mainline)
0430	6919	6D00	0201	7000	4377	0000	70FD	C4[illegible]	[illegible]5E	D003	71FF	C007	D500	0000	74FF	043D	
0440	70FB	4C80	020D	4040	03E5	03E5	7007	437[illegible]	[illegible]0	03E6	03E6	0078	4480	1FFF	C480	0208	
0450	4C18	0462	D008	C012	D008	C480	0208	90[illegible]	[illegible]18	7002	4377	30D0	4377	2000	03E5	0468	
0460	4C80	020D	C002	D0F9	70F7	2010	2000	30D0	C000	9012	4C20	046E	4C80	0468	1801	4C20	
0470	046C	6101	6D00	034B	70F7	C480	0208	D001	4377	0000	4C80	020D	0003	70C4	012C	02D7	I/O AREA
0480	0000	0000	0200	0686	0000	0003	0000	0000	0000	0000	0000	0011	0150	0168	0000	0000	
0490	0001	0000	0000	0001	0001	0000	C000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
04A0	0000	0000	0000	0000	0000	0000	C000	0000	0000	0000	0000	0000	15FF	0040	0040	0040	
04B0	0040	0040	0000	0040	0040	0040	0040	0040	C040	0040	0040	0000	0040	0040	0040	0040	
04C0	0040	0040	0040	0040	0000	0040	0100	00F0	00F0	0100	00F0	00F0	0400	0000	00F0	00F0	
04D0	00F0	00F0	00F0	0400	00F0	00F0	0000	00F0	00F0	00F0	0100	00F0	00F0	0200	00F0	0000	
04E0	00F0	00F0	0602	00F0	00F0	00F0	C0F0	00F0	0000	00F0	00F0	0702	00F0	00F0	00F0	00F0	
04F0	00F0	0000	00F0	00F0	00F0	0702	00F0	00F0	00F0	00F0	0000	00F0	00F0	00F0	00F0	17FF	
0500	0040	0040	0040	C000	0040	0040	0040	00C1	0040	00C3	0040	00C3	0000	0040	00D6	0040	
0510	00E4	0040	00D5	0040	00E3	0000	0040	00E2	0040	0040	00D9	17FF	0040	00C5	0000	0040	
0520	00C3	0040	00C5	0040	00C9	0040	00E5	0000	0040	00C1	0040	00C2	0040	00D3	0040	00C5	
0530	0000	0040	0040	00D9	0040	00C5	0040	0EFF	00D9	0000	0040	00C5	0040	00C7	0040	00C9	

Figure 25. Analysis of a Core Dump (Part 1 of 4)

0540	0040	00E2	0000	0040	00E3	0040	00C5	0040	00D9	07FF	00C3	0000	00E4	00E2	00E3	00D6	
0550	00D4	00C5	00D9	15FF	0000	00D3	00D6	00C3	00C1	00E3	00C9	00D6	00D5	0000	0040	0040	
0560	0040	0040	0040	0040	0040	00C9	0000	00D5	00E5	00D6	00C9	00C3	00C6	16FF	00C9	0000	
0570	00D5	00E5	00D6	00C9	00C3	00C5	0040	00C4	0000	00C1	00E3	00C5	0040	0040	0040	0040	
0580	00C9	0000	00D5	00E5	00D6	00C9	00C3	00C5	17FF	00D5	0000	00E4	00D4	00C2	00C5	00D9	
0590	0040	0040	0040	0000	0040	0040	0040	0040	0040	0040	00C3	00F4	0000	00E2	00E3	00D6	
05A0	00D4	00C5	00D9	0040	03FF	0000	00D5	00C1	00D4	00C5	17FF	00E2	05DD	05E0	05BD	0534	FILE SEQ⊥
05B0	05D7	05CA	0000	02D7	02D9	0005	003C	0000	047E	00E4	0004	5555	0634	0787	4371	05B2	IOD
05C0	C0FA	F01A	4C98	05BD	F017	E014	C002	4480	1FFE	0000	0000	436E	05B2	C0ED	F00C	4C98	
05D0	05CA	F009	E007	D002	4480	1FFE	0000	4C80	020D	0001	7FFF	0FFF	5555	C0FB	D0D8	7004	
05E0	1010	D0D5	7402	020D	436B	05B2	C0D4	4C10	0602	F01F	4C20	05F7	6908	6580	0207	C11E	
05F0	E819	D11E	7406	020D	6500	0000	700B	F011	E0E1	9011	4C20	05FD	E80F	800D	D002	4480	
0600	1FFE	0000	C0B9	D0AB	D400	0201	4C80	020D	4040	FFFF	8000	0017	1000	00D6	4C80	0217	
0610	7001	0000	4368	0000	0368	C400	0219	D400	0208	4C80	0219	6680	0207	6500	0623	6D00	
0620	0207	4E00	0000	0367	063F	02E1	0610	0367	C400	0627	4C28	02F2	E009	D400	0627	C400	
0630	0221	D400	020E	4C00	0291	0000	3FFF	4C00	0628	061B	0637	02E1	0610	0240	5C5C	6600	
0640	0645	6E00	0207	4C00	02A4	05AC	0636	6600	064D	6E00	0207	4C80	0209	6680	0222	6E00	
0650	0209	6600	0624	6580	020A	C102	D201	C103	D202	C480	0207	D203	C006	EA03	D203	4C28	
0660	0628	4C00	02BC	0000	06BE	4365	0000	0368	4366	0011	03E1	4C80	0664	6C00	0207	7428	
0670	0207	6500	022F	6D00	0209	6580	0214	6D00	020A	4C80	0209	6C00	0207	741A	0207	6500	
0680	022F	6D00	0209	6580	0216	6D00	020A	4C8	[illegible]	6C00	0207	740C	0207	6500	022F	6D00	Variable
0690	0209	6580	0226	6D00	020A	4C80	0209	658	[illegible]	C102	D400	020E	7404	020E	C480	020E	Section
06A0	D400	020E	6500	06A8	6D00	020D	4C80	020	[illegible]	020A	C102	D400	020E	7403	020E	C480	(Mainline)
06B0	020E	D400	0201	C101	904E	4C18	06BB	C101	D001	4400	0664	6580	020A	C102	D400	020E	
06C0	C480	020E	D400	020E	6500	06CA	6D00	020D	4C80	020E	6580	020A	C101	D031	C104	902F	
06D0	4CA0	0209	7403	020A	70F5	6102	6D00	0200	6580	020A	C102	D400	020E	7402	020E	C480	
06E0	020E	D400	020E	C480	0208	4C18	06ED	6600	06ED	6E00	020D	4C80	020E	7401	0208	74FF	
06F0	0200	70E6	C480	0208	4C20	06FD	6680	0208	C201	4C98	0207	7401	0208	4C80	0207	0664	
0700	01FE	6580	020A	C102	D400	020E	7402	020E	C480	020E	D400	020E	6600	0712	6E00	020D	
0710	4C80	020E	4C80	0207	6580	020A	C102	D400	020E	7402	020E	C480	020E	D400	020E	6600	
0720	072A	6E00	020D	6600	0729	6E00	0208	4C80	020E	3100	4C80	0209	C400	034B	D400	0350	
0730	C400	034C	D400	0351	1010	D400	0352	4C00	0247	62FE	C600	0352	EE00	034D	D600	0352	
0740	7201	70F8	62FE	C600	0352	4CA0	0209	7201	70FA	4C00	026F	62FE	1010	D600	034D	7201	
0750	70FB	4C00	026F	C400	0349	D40[illegible]	0350	C400	034C	D400	0351	7402	0225	4C80	0225	C400	
0760	0367	4C98	0209	6600	0769	6E00	0208	4C00	06D6	0000	0000	0000	30D0	C400	034B	4C98	
0770	0209	4C00	0714	075F	0664	0422	072C	01FE	0BE7	0739	01FF	0BE7	076D	01FE	0422	074B	
0780	01FF	0BE7	0753	01FE	0BE7	4480	05AE	C400	0213	D400	0207	4C80	0207	4480	05B1	4C00	
0790	0312	0000	697F	6580	1[illegible]F5	7067	0DE3	4C00	088C	7E7E	03E5	0000	6001	1082	74FC	07C0	PRINT⊥
07A0	7033	108E	7420	07C0	7408	07FA	D400	0020	7401	07A7	7029	7401	0027	701B	C045	7001	
07B0	C0EB	6500	0435	71FF	6D00	0028	6129	7058	0799	3200	FF00	0002	0016	0001	FFFF	F100	
07C0	0020	08F6	C0FB	4C08	07CC	C0F6	D034	C0D1	F0F6	4C20	08BC	D0F2	C0CC	1808	E8CA	D0C9	
07D0	4400	090A	6680	07F9	C600	0422	F0C2	80E2	4C02	07DB	88DF	1008	4818	88DE	7201	70BD	
07E0	C012	D0DE	8017	8016	D001	6600	0018	1810	12C0	D480	07A7	C00F	90D0	D00D	C004	D0B7	
07F0	C008	D008	70B8	0020	6000	3480	0030	000C	FFC4	FFC4	FFC4	0000	2000	69B4	6A15	2815	
0800	D899	0874	1005	4C28	07AE	C480	07B2	18[illegible]	4820	700E	086B	1808	4818	7101	7101	C88A	Library
0810	6906	6500	03E6	6600	076B	2001	4C00	04[illegible]	[illegible]4	D0E2	188C	90A1	4C08	07B0	909C	4818	Subroutines
0820	702E	D066	10A0	0852	1808	4C20	0823	C060	D095	7101	690B	7101	6D00	08FC	C0CD	D046	
0830	188C	9059	1802	4C20	07B0	9480	045E	4C10	07B0	D0BF	D0BF	90BC	4C28	07B0	C480	0836	
0840	884A	90B7	D092	C8B2	D0B6	D82E	4400	090A	7401	0027	08A9	7401	0032	1000	70BF	C024	
0850	4830	70FD	0823	180C	4C20	0852	C0A0	D06E	1810	1084	D02C	4C20	0860	1084	4C18	07B0	
0860	9096	4C30	087B	6200	4C10	0878	801D	6201	4C18	0878	8019	7201	4C18	0878	801C	7201	
0870	74FF	08C6	70F9	70BF	0000	0000	2000	3700	C012	1240	E848	D00E	C00A	4C18	080E	C00A	
0880	4C10	0885	0805	70C7	0003	083E	70C4	0001	FFFF	3404	0000	0001	6A30	0878	D074	1001	
0890	4C10	089F	1007	1808	E82E	F0F4	4820	7013	D070	D0F0	082B	74FF	0032	1000	700C	1001	

Figure 25. Analysis of a Core Dump (Part 2 of 4)

08A0	4810	7009	C060	E865	D064	C0E4	90E4	D0E2	4C08	089B	0819	C057	E85C	D05B	4C10	08BC	
08B0	C057	4C20	08C0	C04F	1004	4C10	08C8	C04E	D04F	4050	7401	0027	6600	076B	4C80	0796	
08C0	90CA	D046	70F9	8000	8080	3401	0000	3402	C400	07FB	4C20	07C1	403D	7401	0027	C0A4	
08D0	4C20	08DB	D036	74FA	0875	70E6	082D	74FF	0032	70E2	70E1	74FD	0874	700E	C0AB	4C20	
08E0	08E7	C400	0876	100B	180F	F0A5	D0A3	D09F	C020	4C04	08F8	1801	4C04	08FA	C09B	4818	
08F0	70CB	4C28	0901	08D0	7401	0032	1000	70C4	C008	7001	C089	4480	045F	4C18	08EE	C0C4	
0900	D089	0886	70F1	2000	0004	3440	002E	3701	0000	2000	08CD	10A0	62F8	DE00	0028	7202	
0910	70FC	4C80	090A	62FE	695D	6580	1FF2	6A58	10A0	D02F	C100	18D0	1084	D027	1084	D026	ZIPCO
0920	1010	1084	D024	1084	D023	C101	D00B	C102	D040	C103	D07E	C580	0005	D011	7106	6944	
0930	10A0	C400	0000	18D0	1081	D076	1010	7400	0946	7045	1087	D06E	6580	09AA	C500	0000	
0940	7400	09AC	7007	1008	7006	0000	0000	0000	0000	0000	E062	D05E	C061	7400	0949	1010	
0950	D0F8	C058	7400	0948	7010	7400	0949	7001	700A	D051	74FF	09A9	7018	C480	0969	E052	
0960	E84A	7006	7009	1808	E846	7400	0947	7028	D400	0000	7401	0969	74FF	09A9	7006	6600	
0970	0000	6500	0000	4C00	0000	1010	7400	0946	7003	7400	0949	70B8	7401	0932	70B1	1082	
0980	1005	D02D	1010	1087	4C18	098D	620F	1240	72F9	1000	6A1F	1010	901D	1082	E820	70AB	
0990	18D0	D01F	1010	1083	100D	D019	1083	4C18	09A2	9016	D00F	1010	900D	D00C	6680	09AA	
09A0	C00D	1200	E80C	1806	1086	18D0	C00A	18D0	70BF	0000	0000	0000	0000	FF00	0001	0000	
09B0	000D	0000	00FF	07DB	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	EBPT3
09C0	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	7F7F	
09F0	7F7F	7F7F	7F7F	7F7F	7F7F	647F	257F	267F	677F	687F	297F	2A7F	6B7F	2C7F	7F7F	7F6E	
0A00	7F7F	7F57	7F6D	7F7F	7F15	587F	197F	1A7F	5B7F	1C7F	5D7F	5E7F	1F7F	207F	7F7F	7F62	
0A10	7F23	7F2F	7F7F	7F7F	7F61	7F4C	0D7F	0E7F	4F7F	107F	517F	527F	137F	547F	7F7F	7F16	
0A20	7F7F	7F7F	7F7F	7F7F	497F	407F	017F	027F	437F	047F	457F	467F	077F	087F	7F7F	7F7F	
0A30	7F7F	7F4B	7F4A	7F7F	6979	6580	1FFF	4400	0A97	C074	D400	0B37	C500	0006	4C04	0BD9	SEQOP
0A40	C500	0004	4C08	0BCA	9062	4C30	0BCA	C500	0004	A500	0003	1090	D480	0B38	9058	4C30	
0A50	0BCF	8104	4C08	0BCF	C500	0007	FA00	0B3[illegible]	4C18	0A67	C400	0B35	D10A	C107	F400	0B30	
0A60	4C20	0A82	C105	4C20	0BC0	4C00	0A62	C10[illegible]	[illegible]400	0B34	4C20	0BC5	C400	0B38	8400	0B2F	
0A70	D500	000A	C400	0B2E	D500	0008	6680	0B3[illegible]	[illegible]	D400	0B3A	C400	0AAA	D202	7201	74FF	
0A80	0B3A	70FB	C400	0B35	D500	0000	C500	0001	7401	0B38	D480	0B38	74FF	0B38	8500	0002	
0A90	D500	0002	C015	D500	0009	4C00	0BE2	0AB5	6A13	6B13	6910	7101	6912	C480	0AAB	D400	
0AA0	0001	C500	0006	D400	0B38	4C80	0A97	0140	5656	8017	4040	05E5	0645	1F7E	05AC	05E6	
0AB0	69FD	6580	1FE9	4400	0A97	C0F8	D400	0B37	C500	0009	F0ED	4C18	0AC5	C500	0009	F0E9	SEQIO
0AC0	4C20	0BD4	C0E5	D500	0009	7401	0B38	C480	0B38	74FF	0B38	9102	4C10	0BBB	C10A	4C20	
0AD0	0AD7	C500	0005	4C18	0B20	4C00	0BDE	C500	0008	9500	0003	4C10	0AFE	C500	0007	F052	
0AE0	4C18	0AF4	C500	0004	8500	000A	D000	000A	C500	0008	8043	D500	0008	C500	0007	F041	
0AF0	4C20	0B5B	4C00	0BE2	C500	0005	F03D	4C20	0AE2	C034	D500	0000	4C00	0BE2	C500	0005	
0B00	F032	4C18	0E07	C031	D500	0000	7019	C500	0000	F024	4C20	0B17	C02B	1890	C025	4400	
0B10	00F2	7400	00EE	70FD	C020	D500	0000	7401	0B38	C480	0B38	8012	D480	0B38	74FF	0B38	
0B20	4400	0B92	C500	0005	F00F	4C20	0B5B	C500	0007	F007	4C18	0B44	4C00	0BE2	0001	0002	
0B30	00C9	00D6	00E4	0000	0001	0000	615C	05AC	047E	0000	0000	8010	8011	8012	8013	8014	
0B40	8015	8016	FFFF	0FFF	6680	0B38	C200	D400	0B3A	C400	0AAA	D202	7201	74FF	0B3A	70FB	
0B50	7401	0B38	C480	0B38	80D9	D480	0B38	74FF	0B38	4C00	0BE2	C580	000A	F0D8	4C20	0BE2	
0B60	C0E1	D500	0009	4C00	0BE2	69D1	6580	1FEC	4400	0A97	C109	90CF	4C18	0B8E	C500	0007	SEQCL
0B70	F0C0	4C18	0B7E	C500	0007	F0BC	4C20	0B8E	C500	0000	F0B3	4C18	0B8A	7010	C0B7	D580	
0B80	000A	C500	0008	90AA	A500	0004	1090	80A6	D480	0B38	C0A9	D500	0005	4004	C0B4	D500	
0B90	0009	7050	0B22	7401	0B38	C480	0B38	74FF	0B38	9500	0002	4C10	0BBB	C09A	1890	C500	
0BA0	0005	4400	00F2	C500	0007	F08C	4C18	0BB4	C08F	8085	D500	000A	C081	D500	0008	7400	
0BB0	00EE	70FD	4C80	0B92	C500	0005	F400	0B34	4C18	0BAF	70ED	C400	0B38	D500	0009	7022	
0BC0	C400	0B3C	D500	0009	701D	C400	0B3D	D500	0009	7018	C400	0B3E	D500	0009	7013	C400	
0BD0	0B3F	D500	0009	700E	C400	0B40	D500	0009	7009	C400	0B41	D500	0009	7004	C400	0AA9	
0BE0	D500	C009	6580	0B37	6680	0AAC	6780	0AAD	4C80	0AAF	7F23	7F2F	7F7F	7F7F	7F61	7F4C	
0BF0	0D7F	0E7F	4F7F	107F	517F	527F	137F	547F	7F7F	7F16	7F7F	7F7F	7F7F	7F7F	497F	407F	
0C00	017F	027F	437F	00CC	0000	6935	6A35	6580	0C04	6906	4400	0C27	7400	0032	70FD	C400	RGERR

Library Subroutines

Figure 25. Analysis of a Core Dump (Part 3 of 4)

0C10	0000	D02B	1004	1804	E829	4400	0028	C025	188C	4C18	0C2D	6600	0C39	6A22	0C00	0C40	
0C20	C018	4C18	0C2D	F01E	4C18	0C33	70E8	0000	7101	6D00	0C3A	4C80	0C27	6500	0130	7580	
0C30	007B	4D00	0000	6580	0C3B	6680	0C3C	4C80	0C3A	0000	0000	0000	0000	0000	C000	4C00	
0C40	0000	3A00	0001	0B34	4C20	0BC5	C400	0B38	0000	0B2F	6B6B	6780	1FE6	400E	C200	7201	← RGMV1
0C50	7302	7027	6B63	6780	0000	4006	C302	4C04	0C5A	6873	7303	701D	0C4E	6954	6A55	C300	
0C60	1003	4C10	0C65	7401	0CCB	1001	1804	4C04	0C76	1801	6580	007B	7121	8500	0000	D400	
0C70	0001	C301	D400	0002	4C80	0C5C	7401	0CCA	70F0	6B3E	1808	804C	4C04	0C80	7401	0CCC	
0C80	D400	0003	C100	7400	0CCA	7001	1888	1008	1808	7400	0CCB	7010	7400	0CCD	7033	D200	
0C90	73FF	7001	7019	7201	7400	0CCA	7024	7401	0CCA	1808	1088	70ED	7400	0CCC	7025	1884	
0CA0	E828	D200	73FF	701E	100C	18D0	180C	1084	7400	0CCD	700E	D200	1010	D01C	D01C	D01C	
0CB0	D01C	6500	05AC	6600	0000	6700	1F7E	4C	0615	E80F	70F0	7101	1010	D00C	70C3	1804	
0CC0	1084	7201	E806	70CE	74FF	0CCC	1000	70	0001	00F0	0000	0000	0000	0000	C500	0005	
0CD0	F03D	4C20	0AE2	C034	D500	0000	4C00	0B	10	0005	F032	4C18	0000	C031	6950	6580	← RGMV2
0CE0	1FE3	6A4F	6B50	C100	4C04	0CE7	7002	7401	0D36	188C	4C18	0CFB	904F	4C18	0CF9	904C	
0CF0	4C18	0CF6	7401	0D3A	7401	0D3B	7401	0D39	7002	7401	0D37	1804	108B	6680	007B	7221	
0D00	8600	0000	D400	0002	C101	7102	692F	D400	0001	C100	1808	8030	4C04	0D10	7401	0D38	
0D10	D400	0003	7101	7400	0D37	7001	7002	C101	1884	C100	7400	0D3A	7024	702B	1010	D016	
0D20	C200	1808	1088	D200	7201	1010	D011	7300	7059	D00C	D00C	D00D	D00D	D00D	6500	0773	
0D30	6600	076B	6700	1F7E	4C00	066B	0000	0000	0000	0000	0000	0000	0001	0040	00F0	000F	
0D40	000D	100C	4C18	0D48	1010	D0F4	C100	7001	C0F4	73FF	701E	7400	0D37	7015	7400	0D39	
0D50	7001	701C	7400	0D3B	7009	1884	F0E9	4C20	0D5B	C0E6	7001	C0E3	1084	7010	7400	0D3A	
0D60	700D	E8DC	700B	1884	100C	18D0	180C	1084	70E5	7400	0D37	7004	7001	E0D1	1888	7007	
0D70	7400	0D38	70FA	1884	7101	73FF	1000	7400	0D36	70A4	1888	C200	1008	18C8	7401	0D36	
0D80	D200	70A3	7101	708F	D500	000A	C081	D500	0008	7400	00EE	70FD	4C80	0B92	C500	0005	
0D90	0000	0B34	0091	0091	0091	0091	0091	0091	0091	0091	0091	0000	D819	280E	690F	6A10	
0DA0	6B11	6780	00E4	0816	1002	44A8	002C	6109	080F	1140	4580	0D91	2000	6500	0000	6600	
0DB0	0000	6700	0000	C802	4CC0	0D9B	C009	7004	0000	0300	0000	0F00	00F7	0816	D813	2807	
0DC0	6908	6A09	6B0A	6780	0141	01BB	CDBC	2000	6500	03E6	6600	076B	6700	0141	C803	4CC0	
0DD0	0DBD	407F	0001	9404	0091	0796	0093	D816	280A	690B	6A0C	6B0D	6780	00E4	6102	0810	
0DE0	1140	4580	0DD3	2001	6500	03E6	6600	076B	6700	1F7E	C803	4CC0	0DD6	1804	03E5	0000	
0DF0	0000	030C	0000	4C00	0CDE	0000	4C00	0C4A	0000	4C00	0AB0	0000	4C00	0B65	0000	4C00	
0E00	0A34	0000	4C00	0914	0000	4C00	0792	0000	0000	0000	0000	0000	0000	0C04	09B4	0000	
0E10	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	
0F20	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	FFFF	0000	0000	
0F30	0000	0000	0000	FFFF	0064	0001	0000	0000	0000	0000	0000	0000	003F	0000	0203	0000	
0F40	0000	0000	40D3	0000	0000	0000	0011	0001	0000	0000	0000	0000	0000	0000	0000	0001	
0F50	0001	010A	2000	0000	0000	0000	0001	0000	0000	2D90	0000	0000	0000	0000	2D90	0000	
0F60	0000	0000	0000	02D9	0000	0000	0000	0000	0000	0000	1219	0000	0000	1219	0000	0000	
0F70	0000	0000	01B0	0000	0000	0000	0000	01A8	1000	2000	0000	0000	0000	0000	0000	0000	
0F80	0000	0000	0000	0000	0000	0000	01C0	0000	0000	0000	0000	003F	0000	0000	0000	0000	
0F90	0005	0000	0000	0000	0000	0000	0000	0000	0000	0000	0064	0000	73FF	7012	0000	0000	
0FA0	0011	0000	0001	0006	0000	0000	0000	0001	0001	005C	0000	0002	0000	0008	0000	0028	
0FB0	6161	40E7	C5D8	4040	4040	4040	40D3	4040	4040	4040	4040	4040	4040	40D9	4040	4040	
0FC0	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	4040	
0FD0	4040	4040	4040	4040	4040	4040	4040	4040	0040	0040	0040	0040	0040	0040	0040	0040	
0FE0	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	
0FF0	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	0040	03C0	01BD	
1000	D0A6	4C28	0F86	C400	03C5	D400	0001	1810	4480	03C1	4480	03C1	D100	7101	6680	0FA6	
1010	6316	C600	102F	D100	7101	7201	73FF	70F9	4480	03C1	4480	03C1	4C00	0F86	000C	5C5C	
1020	5CE4	D5C3	D6D9	D940	C5D9	D940	D1D6	C240	E3C6	D9D4	0004	D5D6	E3C5	4040	F710	4040	
1030	4040	4040	4040	4040	4040	C4C9	C1C7	D6D6	E2E3	C9C3	40D4	C5E2	E2C1	C7C5	40C5	E7D7	
1040	D3C1	D5C1	E3C9	D6D5	E240	4C00	109E	4C00	107E	4C00	10FA	74FE	10C2	1000	6580	10C2	
1050	4D00	103E	6580	03C6	712D	6D00	093A	1010	D400	0938	C05D	D066	C05C	D065	C480	10C3	

Library Subroutines

Figure 25. Analysis of a Core Dump (Part 4 of 4)

All data at object time will appear in core storage in the following format:

LLdd	$00Z_1D_1$	$00Z_2D_2$	$00Z_3D_3$	......	$00Z_nD_n$

where LL = eight bits representing n-1 in binary
dd = /FF for alphameric fields, or eight bits representing the number of digits to the right of the decimal point for numeric fields.
n = maximum of 14 words for numeric fields, or maximum of 256 words for alphameric fields.
Z = any hexadecimal quantity from /0 to /F.
D = any hexadecimal quantity from /0 to /F.

- Operations on numeric fields recognize only the D's and consider Z_n as the sign of the number. Z_n = /D is recognized as negative. Z_n = any other hexadecimal number is considered positive. The result of an arithmetic operation (ZADD, ZSUB, ADD, SUB, MULT, or DIV) forces /F for all Z's except Z_n = /D for negative results.

- Numeric literals in the source program are stored with Z's = /F.

- Fields in an I/O area have the following format:

Z_1D_1	Z_2D_2	....	Z_nD_n

The Z_1D_1 may occupy either the first half of a word or the second half of a word.

An exception occurs to this format when P (packed) is specified for a disk file. The following is the "packed" format of a numeric field in the I/O area:

D_1D_2	D_3D_4	.....	D_nZ

Z is the sign of the packed number. The quantity D_nZ may occupy either the first half of a word or the second half of a word. This format requires a leading /0 for padding for fields whose length is even. When this format is "unpacked" to the form previously shown, Z's=/F will be inserted.

READER'S COMMENT FORM

IBM 1130 RPG
Program Logic Manual

Form Y21-0010-0

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

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

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

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

Other ____________________.

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

COMMENTS:

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

YOUR COMMENTS, PLEASE ...

This PLM manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Systems Development Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Programming Publications, Dept. 425

fold fold

IBM 1130 Printed in U. S. A. Y21-0010-0

IBM®

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

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