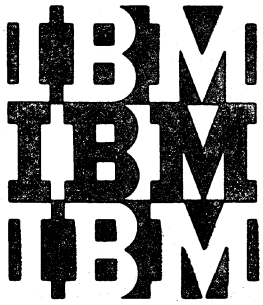
File Number S/360-37
Order Number GY28-6611-0



Program Logic

IBM System/360 Operating System

TESTRAN

Program Logic Manual

Program Number 360S-PT-516

This publication describes the logic of the test translator (TESTRAN) portion of IBM System/360 Operating System. TESTRAN is divided into three parts: the TESTRAN macro instructions, the TESTRAN interpreter, and the TESTRAN editor. The operation of each of these parts is discussed in detail.

Program Logic Manuals are intended for use by IBM customer engineers involved in program maintenance, and by system programmers involved in altering the program design.

PREFACE

This publication describes TESTRAN in three sections, one for each of its three parts. The description is at a sufficiently detailed level to direct the user to the portion of the program listing with which he is concerned.

Before using this publication the reader should be familiar with the contents of the following publications:

IBM System/360 Operating System:

Concepts and Facilities, Form C28-6535

Supervisor and Data Management Services, Form C28-6646

Supervisor and Data Management Macro-Instructions, Form C28-6647

Assembler Language, Form C28-6514

Linkage Editor, Form C28-6538

Introduction to Control Program Logic, Program Logic Manual, Form Y28-6605

First Edition

This edition applies to release 20.1 of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N.Y. 12602

CONTENTS

FIGURES

TABLES

CHARTS

The test translator (TESTRAN) is a program testing and debugging aid for IBM System/360 Operating System assembler language programmers. It consists of three basic parts:

- The TESTRAN macro-instructions.
- The TESTRAN interpreter.
- The TESTRAN editor.

Availability of TESTRAN in the operating system is determined at system generation (SYSGEN) time. Each of the three parts functions during a separate job step within the operating system.

The TESTRAN macro-instructions are expanded at assembly time from their macro-definitions, which are included in the system macro-library at SYSGEN. The status of the macro-instructions, therefore, is that of an expansion of the assembler's macro-instruction library (SYS1.MACLIB). When a user writes TESTRAN macro-instruction statements, they become input to the assembler.

The TESTRAN interpreter functions in the supervisor state at program execution time. Its routines are entered via supervisor call (SVC) interruptions and LINK macro-instructions. The individual routines are reenterable. They are executed "out-of-line" from, but in succession with, the problem program. Some of the interpreter's routines generate test output data.

The TESTRAN editor prints the problem program test output data. Since the editor functions after the execution of the problem program, it is a "post processor." The routines of the editor are serially reuseable. the editor is executed as a separate job step and is treated the same as any processor by the operating system.

THE TESTRAN MACRO-INSTRUCTIONS

The TESTRAN macro-instructions are the user's means of indicating where testing in the problem program is to begin, the exact points at which testing is to take place, and what tests are to be done. Although there are a number of TESTRAN macro-instructions, the TEST OPEN, TEST AT, and GO BACK macro-instructions assume a special significance. The TEST OPEN macro-instruction initiates testing; the TEST AT macro-instruction tells where testing is to take place. (These two macro-instructions need to be coded previous to any other TESTRAN macro-instructions.) The GO BACK macro-instruction is used, either explicitly or implicitly, to return control to the problem program after a series of tests are completed.

THE TIA TABLE

The TESTRAN interpreter action (TIA) table is constructed from the assembler's expansion of the TESTRAN macro-instructions, according to their macro-definitions which are contained in the system macro-library. The table consists entirely of constants.

Each macro-instruction entry into the TIA table has a specific format. (These formats are discussed in detail in Part I of this manual.) The entries in the table are in the same sequence as the source macro-instructions that caused them to be created. Except for the TEST OPEN entry, the TIA table is nonexecutable.

THE TEST OPEN MACRO-INSTRUCTION

The TEST OPEN macro-instruction entry is always the first entry in the TIA table. Although this entry is identical to other macro-instruction entries in that it contains various constants, the TEST OPEN entry has one important distinction. It is the only TIA table entry that is executable. The first byte of each TIA entry is the "type entry" code that is specified in the macro-definition; a different value is inserted for each of the 23 macro-instruction types. Because the type entry byte for the TEST OPEN entry is the operation code for an SVC instruction, this entry is executable.

THE TEST AT MACRO-INSTRUCTION

The TEST AT macro-instruction entry in the TIA table indicates where test services are to be performed in the problem program. At execution time, a TESTRAN interpreter routine inserts SVC instructions into the

problem program at the places specified in the operands of the TEST AT macro-instructions. When the problem program is executed, the inserted SVC instructions cause interruptions that pass control to the TESTRAN interpreter which, in turn, initiates the performance of the test services.

THE GO BACK MACRO-INSTRUCTION

The GO BACK macro-instruction entry in the TIA table indicates when control is to be returned from the TESTRAN interpreter to the problem program. Since all testing takes place in the supervisor state, it is necessary to return control to the problem state before problem program execution can be continued.

The GO BACK macro-instruction may be coded, calling for its functional routine explicitly, or the routine may be called for implicitly. Regardless of how control is passed to the go back routine, its execution allows control to be returned to the problem program so that it may resume execution.

THE TESTRAN INTERPRETER

After assembly, the problem program and associated TIA tables must be linkage edited into a single load module. The load module produced is neither reenterable nor reuseable. When this load module is specified in an EXEC statement with a TEST parameter, the control program prepares the job step for execution. Both the problem program and the TESTRAN interpreter routines that perform the tests execute under control of the same task control block. The TESTRAN interpreter routines generate test output data for processing and printing by the TESTRAN editor in a subsequent job step. Testing of the problem program occurs in succession with the execution of the problem program, but out-of-line from it; i.e., the test service routines of the interpreter are executed at the points within the problem program where the user has indicated he wanted them. The requested series of tests are performed, while the problem program execution is temporarily suspended, and at the conclusion of the series of the test services, problem program execution is resumed.

OPERATION OF THE INTERPRETER

At problem program execution time, control must be passed to the TEST OPEN entry in the TIA table to initiate testing. Since that entry is an SVC instruction, its execution causes an interruption that passes control to the TESTRAN interpreter.

The interpreter inserts a TESTRAN SVC into the problem program at each point specified in the TEST AT macro-instructions. The two bytes of the problem program displaced by the SVC instruction are stored in an interpreter table for later retrieval and execution.

Once the SVC insertions have been made, control passes to the entry point specified for the problem program, and execution of the problem program begins. When an SVC instruction is encountered in the problem program, the interruption processed for the SVC causes control to be passed to the interpreter's router routine, which determines the TEST AT macro-instruction that caused the interruption and analyzes the TIA table entry following that TEST AT entry. The router then passes control to the proper service routine for the performance of the requested test service. When the current series of test requests in the TIA table is completed, the router routine passes control to the go back routine. The go back routine retrieves the two bytes of displaced problem program instruction from the interpreter table where they were stored when the TESTRAN SVC was inserted, reassembles the instruction at a remote location and executes it. It then returns control to the problem program. When the problem program terminates, the function of the TESTRAN interpreter is completed.

The test output data, generated as a result of the execution of the interpreter test service routines is written onto a system data set whose ddname is SYSTEST. This data is stored for later retrieval and editing by the TESTRAN editor.

THE TESTRAN EDITOR

The TESTRAN editor transcribes the information contained in the test output data created by the interpreter into a printable output. It is a post processor in that it functions only after the problem program whose test output it is to edit is terminated. The editor consists of discrete routines that read the test output data generated by the TESTRAN interpreter and select the records with the proper output selection codes for processing. The

records are transcribed into the correct output format as determined by the type of interpreter routine that generated the data. Proper headings for the record type and available symbolic labels are written with the data onto the system print data set (SYSPRINT).

The editor analyzes the output selection code associated with a test output data record to determine whether or not it is to process that particular record. This determination is based upon the output selection code or codes specified in the job control language execute (EXEC) statement for the editor job step. The output selection code indicated there is compared with the output selection code in the record itself; those records whose output selection codes are acceptable are processed, and those whose output selection codes have not been specified are skipped. The actual output selection code for the job step is an integer from one to eight, a blank, or the letter A. The output selection code in the record itself is either a value (see Table 1(C)) or a hexadecimal zero.

The TESTRAN editor is always a separate job or job step. It is not automatically executed, but must be called for in the same manner as any other processor.

RELATIONSHIP BETWEEN TESTRAN AND THE OPERATING SYSTEM

The three parts of TESTRAN as they operate within the operating system are shown in Figure 1. Parameters and options within job control statements are coded by the user to initiate the use of TESTRAN in a specific job. Control statements initiate the assembly of the source statements, problem program and TESTRAN interpreter execution, and operation of the TESTRAN editor.

Job control language statements calling for assembly require the reading of the TESTRAN source statements and an expansion of them. The expansion is directed by the macro-definitions in the macro-library. The expanded macro-instructions are assembled into the (TIA) table in the form of a control section separate from the problem program. The TIA table and the problem program text are written onto an output data set as object modules.

The symbol table and external symbol dictionary from an assembly with a test parameter are written out with the object modules in an 80-character record format labeled SYM. The TESTRAN interpreter saves and test open routines together locate and copy the data onto SYSTEST. The TESTRAN editor uses the data to symbolically label dumps and to identify section definitions in its storage map. At this point, the user either exits and suspends further operations, or supplies job control statements to call for the execution of the problem program.

Before the object modules can be executed, they must be linkage edited into a single load module and fetched into main storage. At appropriate times during the execution of the problem program, the TESTRAN interpreter honors the test requests that have been coded into the TIA table. Because the interpreter operates wholly in supervisor state, and the problem program operates entirely in problem state, operation alternates between the two states on a schedule determined by the TESTRAN statements that are being interpreted.

Test output data generated by the interpreter routines is written onto SYSTEST for later retrieval and editing by the TESTRAN editor. The functions of the interpreter are completed when the problem program being tested reaches an end-of-task.

When job control language statements calling for the TESTRAN editor are supplied by the user, the editor is fetched for execution. This may be directly following the execution of the problem program or at some later time. The editor reads the interpreter's test output data from SYSTEST and determines from the output selection code whether a record is to be processed or skipped. If the record is to be processed, the editor provides proper headings, applies symbolic labels (when the symbol table is provided), converts data to printable format, and writes the record onto SYSPRINT.

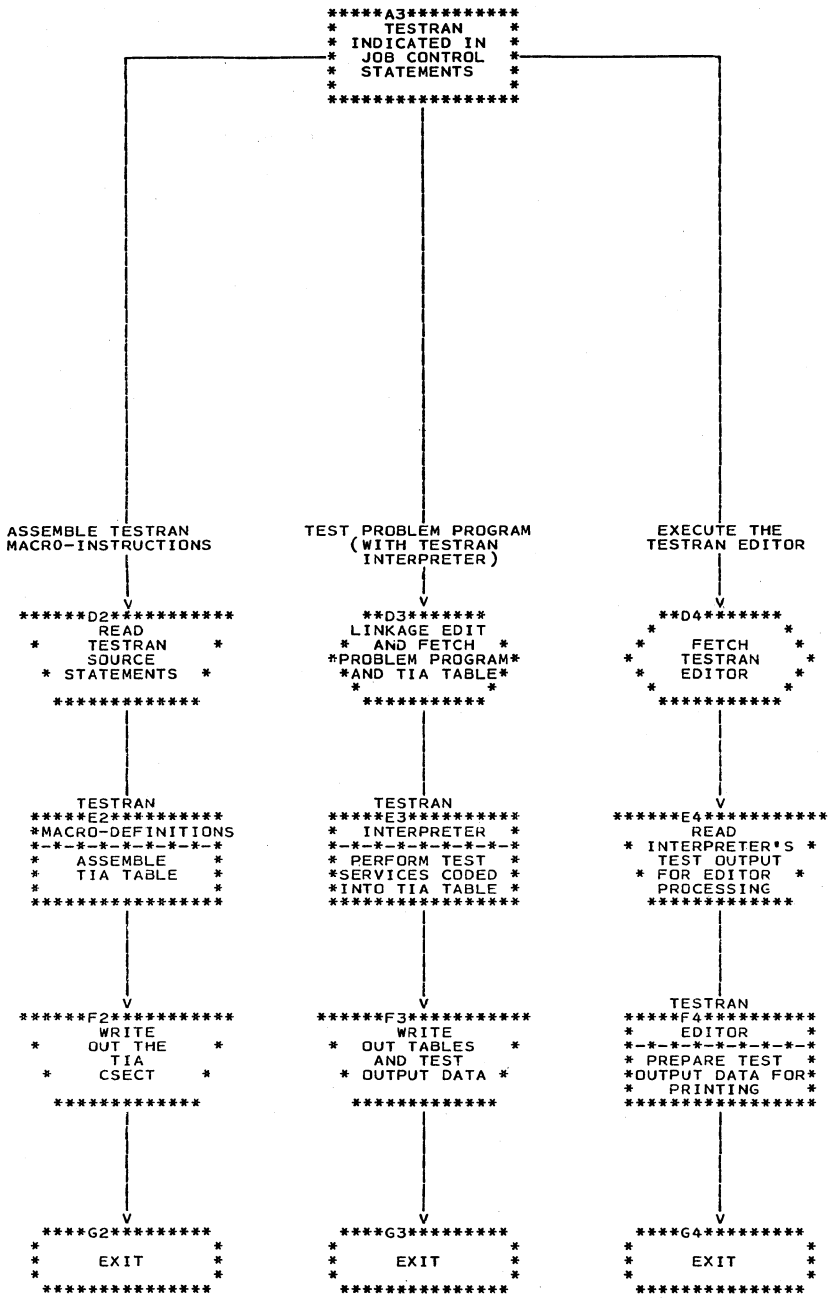


Figure 1. Three Parts of TESTRAN Within Operating System/360

The TESTRAN macro-instructions are a part of the macro-library of the IBM Operating System/360 assembler processor. There are five basic macro-instructions in TESTRAN: TEST, SET, GO, DUMP, and TRACE. By including the first operand, these five basic macro-instructions are extended into 23 usable macro-instructions. An installation that uses TESTRAN enters the macro-definitions for these macro-instructions into the assembler macro-library at SYSGEN time.

The TEST, SET, and GO macro-instructions are classed as "test control" macro-instructions because their expansions provide internal control of TESTRAN. They do not generate output as a primary purpose. The function of the DUMP and TRACE macro-instruction expansions is the generation of test output data. These two macro-instructions are classed as "test action" macro-instructions.

System macro-instructions normally expand into a sequence of executable instructions. TESTRAN macro-instructions, however, are translated and assembled into a series of constants called the TIA table. These entries in the TIA table are test requests to be honored by the TESTRAN interpreter.

Because the assembler does not expand any other TESTRAN macro-instruction until after a TEST OPEN macro-instruction is expanded, TEST OPEN must always be the first macro-instruction in an assembly.

THE TIA TABLE

The TIA table consists entirely of constants. For each type of TESTRAN macro-instruction, there is a specific format that includes some mandatory and some optional fields. The formats are shown in Figure 2. For a detailed presentation on how to code these macro-instructions, refer to the publication IBM System/360 Operating System: Control Program Services. Each entry in the table has a type-entry field, an identification field, and a length field. The three fields are referred to collectively as the "common data." The other fields of an entry are referred to collectively as the "variable field." Within the variable field there are various address (A) fields and modifier fields. Specific formats for each TESTRAN macro-instruction type are found in Appendix A.

Except for the TEST OPEN entry whose first two bytes form the operation code for an SVC instruction, the TIA table is composed of nonexecutable code. The TEST OPEN entry permits control to be passed to the TIA table and an interruption generated to allow testing to be initiated.

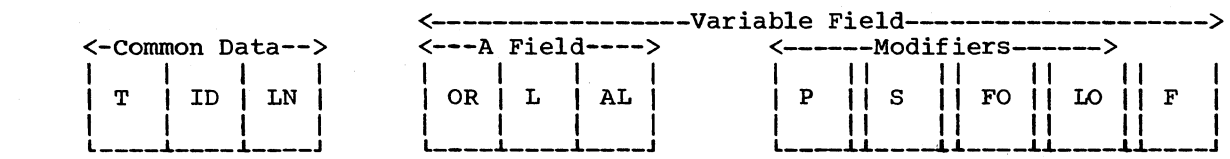
ASSEMBLING THE TIA TABLE

TESTRAN macro-instruction statements may be coded and assembled with the problem program assembler-language source statements, or they may be coded as a separate assembly. The TESTRAN macro-instructions are always assembled into a separate control section (CSECT). Depending on how the user wants to separate his test sequences, the TESTRAN macro-instructions may be assembled into more than one CSECT. However, a TEST OPEN macro-instruction must be written for each CSECT desired. It is the CSECT, which has been formed for a series of TESTRAN macro-instructions, that is referred to as the TIA table.

Assembling the TIA table in a separate CSECT from the problem program permits the linkage editor to delete the TESTRAN CSECT (TIA table) from a load module when testing has been completed without the necessity of reassembling the problem program. If additional or different tests are desired, they can be coded into the TIA CSECT without altering the problem program being tested. Eliminating the need for reassembly of the problem program eliminates the possibility of new errors being introduced in the assembly process.

When the assembler processor encounters a macro-instruction statement, it expands the statement into assembler language statements as directed by the corresponding macro-definition. The assembler language statements generated during the expansion of TESTRAN macro-instructions are DC (define constant) statements.

After the assembler has generated the appropriate DC statements for a given TESTRAN macro-instruction, it translates them into the machine language constants that form the TIA table entry for the macro-instruction. The particular constants generated for a given TESTRAN macro-instruction are determined by its corresponding macro-definition and by the specific operands coded for the macro-instruction statement.



Legend

T specifies type byte. The 1-byte hexadecimal code indicates the macro-instruction type this entry represents. (See Table 1 (A).)

ID represents identification byte. This 1-byte number is assigned to this macro-instruction at assembly time. (SVC number is assigned if type is TEST OPEN.)

LN indicates entry length byte. This 1-byte number specifies the total length (in bytes) of this TIA entry.

Note: Multiple A fields are possible.

OR specifies organization byte. This 1-byte code indicates the organization for this A field. (See Table 1 (B).)

L denotes length byte. This 1-byte number indicates the length (in bytes) of the AL field if it is a literal.

AL indicates address/literal. This is a variable length field whose contents depends on the organization byte.

P specifies output selection code. This 1-byte code represents the 1-to-8 priority (output selection code) scheme used by the TESTRAN editor. (See Table 1 (C).)

S represents scale. This 2-byte count specifies the number of places (binary or decimal) point is to be shifted left.

FO indicates format. This 1-byte data format identifier code specifies the format of the data to be referred to by this entry. (See Table 1 (D).)

LO specifies length overrider. This 2-byte value overrides the otherwise specified length of the data fields referred to by this entry.

F represents flag byte. This 1-byte value indicates the presence of the P, S, FO, and LO modifiers in the variable field.

Figure 2. General Format for TIA Table Entries

Table 1. Explanation of General Format Symbols for TIA Table Entries

A. Type Byte Code

Hex Code	Entry Type
02	TEST AT
06	DUMP DATA
0A	TEST OPEN
0E	DUMP CHANGES
12	DUMP MAP
16	DUMP PANEL
1A	DUMP COMMENT
1E	DUMP TABLE
22	TRACE REFER
26	TRACE CALL
2A	TRACE FLOW
2E	TRACE STOP
32	TEST CLOSE
36	GO IN
3A	GO OUT
3E	GO BACK
42	TEST DEFINE COUNTER
46	TEST DEFINE FLAG
4A	SET COUNTER
4E	SET FLAG
52	SET VARIABLE
56	TEST ON
5A	TEST WHEN
5E	GO TO
62	END TABLE MARKER

B. Organization Byte Code

Bit 0: "L" Field Flag
- 1 - Present
- 0 - Absent

Bit 1: Not Used

Bits 2 and 3: AL Field Type
- 00 - Literal
- 01 - 24-bit absolute address
- 10 - 16-bit displacement address
- 11 - 1-byte register number (contents used as a value, not an address)

Bits 4 through 7: Index Register
- 00 - No indexing
- 01 through 15 - General register to be used to index address in AL field

C. Output Selection Code

80	Output Selection 1
40	Output Selection 2
20	Output Selection 3
10	Output Selection 4
08	Output Selection 5
04	Output Selection 6
02	Output Selection 7
01	Output Selection 8

D. Format Code

04	Fixed-point half
08	Fixed-point full
0C	Floating-point long
10	Floating-point short
14	Packed decimal
18	Unpacked decimal
1C	Character
20	Address
24	Hexadecimal
28	Instruction
2C	Binary

TESTRAN MACRO-DEFINITIONS

There is a macro-definition for each of the five basic TESTRAN macro-instruction types. Macro-definitions are placed in the macro-library at SYSGEN time. Brief descriptions of the five macro-definitions follow.

TEST MACRO-DEFINITION

When the assembler encounters a TEST macro-instruction, it refers to the TEST macro-definition for directions. The TEST macro-definition directs the assembler to check the macro-instruction statement's operand fields for critical errors. (A critical error is one that makes further processing of the macro-instruction impossible.) If a critical error is detected, an error message is written, and expansion of the macro-instruction is terminated. If no critical errors are detected, the common data fields are generated, and positional operands are processed sequentially from left to right. Following the processing of positional operands, any keyword operands present are processed, the necessary flags are generated, and expansion of the macro-instruction is completed.

The flow of the TEST macro-definition process is shown in Chart 10.

SET MACRO-DEFINITION

When the assembler encounters a SET macro-instruction it refers to the SET macro-definition for directions. The SET macro-definition directs the assembler to perform the necessary diagnostic functions. If a critical error is detected, an error message is written and expansion of the macro-instruction is terminated. Otherwise, the common data is generated and positional operands are processed sequentially from left to right. If any keyword operands are present they are processed last.

The flow of the SET macro-definition is shown in Chart 11.

GO MACRO-DEFINITION

When the assembler encounters a GO macro-instruction, it refers to the GO macro-definition for directions. The GO macro-definition directs the assembler to check for critical errors in the operand fields. If a critical error is detected, an error message is written and expansion of the macro-instruction is terminated. If not, the common data is generated, then the necessary addresses and flags are generated to complete the expansion.

The flow of the GO macro-definition is shown in Chart 12.

DUMP MACRO-DEFINITION

When the assembler encounters a DUMP macro-instruction, it refers to the DUMP macro-definition for directions. The DUMP macro-definition directs the assembler to check for critical errors in the operand fields. If a critical error is detected, an error message is written and expansion of the macro-instruction is terminated. If none are detected, the common data is generated, and the positional operands are processed sequentially from left to right. Following this, keyword operands present are processed, and any necessary flags are generated to complete the expansion.

The flow of the DUMP macro-instruction is shown in Charts 13 and 14.

TRACE MACRO-DEFINITION

When the assembler encounters a TRACE macro-instruction, it refers to the TRACE macro-definition for directions. The TRACE macro-definition directs the assembler to check for critical errors. If any are detected, an error message is written and expansion of the macro-instruction is terminated. Otherwise, the common data is generated and the positional operands are processed. Following this, the keyword operands present are processed and flags are generated to complete the expansion.

The flow of the TRACE macro-definition is shown in Chart 15.

The TESTRAN interpreter consists of the 25 setup and service routines that are used to test the problem program. With the exception of the save routine, all of the routines function in the supervisor state at problem program execution time.

The 16 service routines are executed in response to a given macro-instruction and, since macro-instructions can be coded in a variety of sequences, the service routines can also be executed in the same variety of sequences.

The 9 set up routines are executed at specific points in the execution of the problem program. They support the service routines by acquiring storage for tables and by establishing the flags and switches needed by the service routines. Entry into the various setup routines is gained through the execution of TESTRAN SVC instructions and through the execution of the system macro-instructions LINK and XCTL. Exit from the routines is via a RETURN or an XCTL macro-instruction.

Except for the tracer routine, which normally is entered via an XCTL macro-instruction from the go back routine, all service routines are entered from the router (setup) routine via a LINK macro-instruction. Table 2 is a complete list of the interpreter modules, their classifications, and their normal residences.

The TESTRAN interpreter generates test output data and writes it into SYSTEST. This test output data consists of two physical records for each logical record. The physical records include a prologue

Table 2. TESTRAN Interpreter Modules

Module Designation	Module Residence	Service or Setup	Macro-Instruction Interpreted
IEGTTRNA	L	S	DUMP DATA,DUMP CHANGES
IEGTTRNB	L	S	DUMP COMMENT
IEGTTRNC	L	S	DUMP PANEL
IEGTTRND	L	S	GO TO,GO IN,GO OUT
IEGTTRNE	L	S	TEST ON
IEGTTRNF	L	S	DUMP TABLE
IEGTTRNG	L	S	TEST WHEN
IEGTTRNH	L	S	TEST CLOSE
IEGTTRNJ	L	S	GO BACK
IEGTTRNK	L	S	DUMP MAP
IEGTTRNL	L	S	TRACE FLOW,TRACE CALL,TRACE REFER
IEGTTRNM	L	S	TRACE STOP
IEGTTRNN	L	S	SET COUNTER
IEGTTRNO	L	O	none (this is overlay 1 routine)
IEGTTRNP	L	S	SET FLAG
IEGTTRNR	L	S	SET VARIABLE
IEGTTRNT	L	S	TRACE FLOW,TRACE CALL,TRACE REFER
IEGTTRNX	L	O	none (this is overlay 2 routine)
IEGTTRNZ	L	O	none (this is trace interrupt routine)
IEGOPEN2	L	(O&S)	none (this is phase 2 of TEST OPEN)
IEGOPEN3	L	(O&S)	none (this is phase 3 of TEST OPEN)
IEGTTROT	L	O	none (this is router routine)
IEC038	R	O	none (this is resident SVC routine)
IGC004I	I	(O&S)	TEST OPEN (this is phase 1 of 3)
IGC0106A	I	O	none (this is the save routine)

Legend
L - Module resides or Link library
R - Module is part of resident nucleus
I - Module resides on SVC library
S - Module is service, interprets named macro-instruction
O - Module is setup, supports service modules
(TEST OPEN routines are a combination of both types)

record, which has a fixed length of 128 bytes, and a data record, which has a variable length. The information concerning which macro-instruction caused the output to be generated is contained in the prologue record.

Although the test control macro-instruction entries (i.e., TEST, SET, and GO) do not generate output records as their primary function, the TESTRAN interpreter posts the execution of these macro-instruction entries in a buffer. When a test action macro-instruction entry (i.e, DUMP or TRACE) generates an output record, this buffer is written onto the prologue portion of that test action record and the buffer is cleared. Because of the size of the prologue record, a maximum of 28 test control macro-instruction entry executions can be recorded on it. The data record of the record pair is of variable length, dependent on the record type.

THE SETUP ROUTINES

The setup routines perform the "housekeeping" and selection functions necessary to support and control the choice of the service routines. These routines are executed at specific times, relative to the execution of the problem program. Their sequence of execution is restricted, certain routines being prerequisite to others.

The save routine must be the first setup routine executed. It is executed when the test attribute of the load module is recognized by contents supervisor. Should the execution of a test open routine be attempted prior to that of the save routine, the test open routine will not be executed. Instead, control will be given to the problem program and the problem program will be executed without being tested.

If the save routine is executed prior to the time the test open routine is given control, however, the test open routine inserts SVC instructions into the problem program. These SVCs generate interruptions in the problem program and pass control to the TESTRAN interpreter.

In an overlay program, the overlay supervisor calls the TESTRAN overlay routine whenever it fetches a new segment into main storage. Once the test open routine in executed in the root segment, this overlay routine performs test open functions on the newly loaded segment.

The resident SVC routine is a portion of the control program nucleus. When an interruption is caused by one of the SVC instructions inserted by the TEST OPEN routine, this routine passes control to the router routine.

The router routine can be executed only after the save and test open routines have been executed, control has passed to the problem program, and an inserted SVC instruction has been executed. The trace interrupt routine is given control when an interruption occurs during a trace.

THE SAVE ROUTINE (IGC0106A)

The TESTRAN save routine (see Chart 30) retains the address and the extent data used by program fetch when load modules are acquired from their external residence. The test open routine later uses this data to acquire the symbol table and ESD records from the external residence.

Entry to the TESTRAN save routine is via an XCTL from the TSO TEST save routine (IGC0006A) (see TSO Test/TMP PLM). If the TESTRAN flag (test attribute bit) is set in the partitioned data set directory entry for the program load module member, the contents supervisor issues an SVC instruction. This instruction results in a load of the TSO TEST save routine. The TSO TEST save routine will XCTL to the TESTRAN save routine if the TCB under which the SVC was issued does not indicate a TSO task.

The TESTRAN save routine is executed after the load module containing the problem program and the TIA table is fetched, but before it is given control.

This routine acquires areas of main storage for the DCB/REL core and control core tables. It enters data into the DCB/REL core that relates to the external residence (disk) addresses of the symbol table and the external symbol dictionary (ESD) from the assembler, and the composite external symbol dictionary (CESD) from linkage editor. (The dictionaries and the table are generated at assembly time.) It also fills in a loaded address field in DCB/REL core for each CSECT in the load module and copies of the data control block (DCB) and the data extent block (DEB) that were used by program fetch in loading the module. Addresses in the DCB and the DEB are corrected to show the actual loaded address. (For a detailed description of DCB/REL core, see Appendix B.)

As one of its fields, control core contains a pointer to the DCB/REL core. It also contains fields for the pointers that are used by the TESTRAN interpreter to refer to its control blocks, and fields for the various flags, buffers, temporary storage areas, counters, and control blocks that are necessary for effective interface with the basic sequential access method (BSAM) routines. The address of the control core table is entered into the task control block (TCB) for the task.

Only a few of the fields in control core are filled at save time from the data supplied by the save routine. Data from the problem program or the TIA table is used to complete other fields during execution of the test open and router routines.

THE TEST OPEN ROUTINES

The test open routines are the test routines that perform setup functions for the TESTRAN interpreter. The other test routines are discussed under "The Service Routines."

Three routines are used to perform the TEST OPEN functions: TTOPEN 1, TTOPEN 2, and TTOPEN 3. These routines can be entered in two ways: one in which control is initially passed to the TEST OPEN entry in the TIA table at execution time, and the second in which a direct branch to the TEST OPEN entry is contained within the problem program. When control is passed to the TEST OPEN entry, the SVC 49 that is the first two bytes of the entry is executed. The interruption processed as a result of the SVC passes control to the TTOPEN 1 routine.

In the first method of entry, the user supplies a linkage editor control ENTRY statement to the linkage editor at linkage edit time. This statement, when properly coded to do so, overrides the END statement in the user's assembler source module and causes control to be initially passed to the instruction named as the operand in the ENTRY statement. For example, if the user codes END START in his source module, at execution time control normally passes to the instruction named START to begin execution of the program.

When a linkage editor control ENTRY statement such as ENTRY TEST is used, however, control passes to the instruction named TEST. Should TEST be the name of the TIA table associated with the problem program (and the name of its TEST OPEN entry) control passes to the TEST OPEN entry while the problem program still has its original END START statement (now overriden) for use after testing is deleted.

To allow the test open routines to pass control to the beginning of the problem program so that it may be executed, the user must also code within his TEST OPEN macro-instruction an "entry" positional operand that specifies the normal program entry point (i.e., START). In this entry method, deletion of the TESTRAN CSECT from the load module is accomplished by a pass through the linkage editor. Altering or reassembling the problem program after testing is completed is not necessary.

In the second method of entry, the user supplies a direct branch from within the problem program to the TEST OPEN entry in the TIA table. Reassembly of the problem program at the completion of testing in order to delete the branch is necessary when this method of entry is employed. Except for the reassembly requirement, this method has the same operation and requirements as the first method.

If the problem program to be tested is an overlay program and has multiple TIA tables, the test open routine ignores listed TIA tables which are not in main storage.

TEST OPEN macro-instructions associated with segments other than the root segment are called "secondary TEST OPEN macro-instructions," and as such do not require the inclusion of "entry" positional operands in their macro-instruction statements.

Secondary TEST OPEN macro-instruction entries contained in the TIA tables associated with overlay segments should never be given control and, therefore, never executed. Instead, the overlay 1 routine performs the necessary test open functions, such as inserting SVC instructions into the newly loaded segment. In an overlay program, the test open routines execute only once per execution of the program.

The TTOPEN 1 Routine (IGC0004I)

The TTOPEN 1 routine (see Chart 31) is entered initially as a result of the execution of the TEST OPEN entry in the TIA table (the first two bytes of which are the operation code for an SVC instruction). This routine determines if the save routine has been executed. If the save routine has not been executed, the test open functions are not performed, and control is returned to the problem program. The problem program then executes as though there were no TESTRAN macro-instructions coded to test it (i.e., no testing takes place).

If the save routine has been executed, the TTOPEN 1 routine loads the router routine into main storage and proceeds to examine the TEST OPEN entry in the TIA table to establish the maximums to be used for this execution of the problem program (i.e., number of lines of test output data allowed and number of TESTRAN statement executions allowed). This information, the output selection code, and the link or load

option specified in the TEST OPEN entry are placed into the proper fields of control core. (For a detailed description of control core, see Appendix B).

At this point in its execution, the TTOPEN 1 routine links to the TTOPEN 2 routine if it determines that this is its first execution since the execution of the save routine. Otherwise, the TTOPEN 1 routine links to the TTOPEN 3 routine to prepare the problem program for testing.

After the TTOPEN 3 routine has operated, inserting SVC instructions into the problem program and establishing various TESTRAN tables, it returns control to the TTOPEN 1 routine. The TTOPEN 1 routine writes a map change record (if necessary), the TIA and reference tables, and a test open record onto the intermediate data set (SYSTEST) for use by the TESTRAN editor. The map change record is generated for each program overlay and it indicates the presence and active or inactive status of each program segment in main storage.

Note: The link or load mode option is specified in the TEST OPEN macro-instruction. In the load mode, routines are acquired from the link library via the LOAD macro-instruction and then control is passed to them via a LINK macro-instruction. Routines, up to the available limits, are retained in storage. In the link mode, routines are acquired from the link library when they are needed. No attempt is made to retain them in storage.

The TTOPEN 2 Routine (IEGOPEN2)

The TTOPEN 2 routine (see Chart 32) performs the copying functions of test open. With this routine, control core fields are initialized and the data whose address and extent has been recorded and retained in DCB/REL core by the save routine is copied for later use by the TESTRAN editor.

The TTOPEN 2 routine uses DCB/REL core address data to read records (symbol table, ESD, and CESD) from their external residence into an area of storage it acquired for a buffer. It copies them from the buffer onto SYSTEST. Buffers acquired by the routine and storage acquired for use as DCB/REL core are then released. Control returns to the TTOPEN 1 routine.

The TTOPEN 3 Routine (IEGOPEN3)

The TTOPEN 3 routine (see Chart 32) prepares the problem program for testing. It first determines if the TIA table it refers to is already open, and if so, returns control to the TTOPEN 1 routine. If the TIA table is not open, this routine tests for valid entry types in the TIA table.

If all the entries are valid, the routine computes the amount of storage it will need for a block called table core by scanning the entire TIA table to determine the number of flags and counters (either explicitly or implicitly defined) and the number of TEST AT entries in the table. (For a description of table core, which includes the reference, counter, and flag tables, see Appendix B.) The routine then acquires the necessary main storage. The routine scans the TIA table a second time and inserts SVC instructions into the problem program at each point specified in a TEST AT entry in the table. It also inserts, into the reference table area that corresponds to that TEST AT entry, the two bytes of problem program code that the SVC instruction displaced, and the locations of the SVC instruction in the problem program and the TEST AT entry in the TIA table.

Likewise, the TTOPEN 3 routine enters a flag into the flag table area for each flag specified in a TEST DEFINE FLAG TIA entry. The counter table area of table core is filled in, in this same manner, with the counters specified in the TEST DEFINE COUNTER TIA table entries. Pointers to the first entry in each of the reference, flag, and counter tables are entered into the fixed area of table core (or the previous reference, flag, and counter tables). A pointer to table core is entered into control core (or the previous control core).

To ensure that the TIA table associated with a problem program is not altered by that program, the TESTRAN interpreter computes a "checksum" of the TIA table and stores it in table core.

Note: The checksum is a summation of the entire TIA table, added in 4-byte increments. The add logical instruction is used to compute the 4-byte sum. When the length of a TIA table is not an even multiple of four bytes, the remaining bytes are right justified and added to the low order end of the previous sum.

Whenever the TESTRAN router routine is entered during execution of the problem program, a checksum of all TIA tables in

main storage as they exist at that time is computed. If a newly computed checksum does not match the one stored in table core, that TIA table has been altered and a message to indicate this is written onto SYSTEST. As a result the task is brought to an abnormal end (ABEND).

If additional TIA tables are listed in an "OPTEST=" operand field in the TEST OPEN entry, the next TIA table is examined to determine whether or not it is currently in storage. If it is, that TIA table is also opened (i.e., the TTOPEN 3 routine is repeated for the next TIA table). If there are no more TIA tables to be opened, control is returned to the TTOPEN 1 routine.

THE RESIDENT SVC ROUTINE (IGC038)

The resident SVC routine (see Chart 30) consists of the parameters and instructions for linking to the router routine. When a TESTRAN-inserted SVC instruction is executed during problem program execution/testing, the operating system's SVC handler passes control to the resident SVC routine. This routine, in turn, passes control to the router routine that was loaded by the TTOPEN 1 routine. When a series of TIA table entry interpretations are complete, control is returned from the router routine, through the resident SVC routine, to the problem program.

THE ROUTER ROUTINE (IEGTTROT)

The router routine (see Chart 33) determines from the entries in the TIA table which of the TESTRAN interpreter service routines are to be executed. Each time it is entered, this routine tests the validity of each TIA table currently in main storage by computing a checksum of the table and by comparing that checksum with the original one computed and stored in table core by the TTOPEN 3 or overlay 1 routine. If the checksums do not agree, the router routine issues an error message to indicate that the TIA table has been altered and brings the task to an abnormal end (ABEND).

Some of the service routines of the interpreter are classified as "test control" routines because they are not concerned with the generation of test output data, but rather with the internal control of testing. As one of these routines is executed, the router routine enters information to indicate that it has been executed into a prologue record buffer. This information, in the form of a list of executed statements, is written in the prologue record of the output record pair generated for the next "test action" routine executed.

To pass control to the proper service routine, the router routine first determines the address of the SVC instruction that caused it to receive control by subtracting two bytes (length of the SVC) from the address of the next problem program instruction as it appears in the old program status word (PSW). This gives the routine the address of the SVC instruction in the problem program. The router routine then scans the reference table to find a matching address in the object program address field. When it finds this matching address, the address in the TIA address field (within the same entry in the reference table) is the address of the TEST AT macro-instruction entry that caused the SVC to be inserted. The router routine then scans the TIA table to find the type byte of the next entry.

The router routine uses the type byte as an index factor to find the correct character in an internal table. The character is appended to the symbolic name root (IEGTTRN) of the service routines to complete the symbolic name of the required service routine. The router routine then links, or loads and links, to the routine. Control returns to the router routine when the service routine has completed its functions. If the previous routine was not a go back routine, the router routine examines the next entry and repeats the above procedure. If it was a go back , control is returned to the problem program or to the tracer routine, as explained in the go back discussion.

Two of the router's subroutines are contained entirely within it and are branched to as required. The first, HBADDRSR (see Chart 34), interprets "A" field expansions (see Appendix A) in the TIA entries, computes an actual address for these entries, and returns that address to the service routine that needs it. The second, HQOUTPUT (see Chart 34) controls the writing of prologue and data records of an output record pair. It also checks the output-line count to ensure that the number of lines generated has not exceeded the maximum allowed.

THE OVERLAY ROUTINES

The Overlay 1 Routine (IEGTTRNO)

In an overlay program, whenever the overlay supervisor causes a new segment with test attributes to be fetched into main storage it links to the overlay 1 routine (see Chart 35).

The overlay 1 routine first determines whether or not the test open routine has operated, and if it has not, this routine causes a map change record to be written onto SYSTEST and returns control to the overlay supervisor.

If the test open routine has been executed, the overlay 1 routine next checks the count of TIA tables listed for the OPTEST= operand of the TEST OPEN entry in the root segment TIA table. If the OPTEST count is zero, and if the trace mode switch is on, control is passed to the overlay 2 routine. For OPTEST count zero and trace mode switch off, a map change record is written onto SYSTEST and control is returned to the overlay supervisor.

When the OPTEST count is non-zero, the overlay 1 routine determines which TIA table is next in the list and if that table has ever been opened. If the TIA table has previously been opened, this routine checks to find whether it is now in main storage. If the TIA table is not in main storage, its associated reference, flag, and counter tables are flagged inactive, and the OPTEST count is again tested for zero or nonzero.

If the next TIA table has been opened previously, and is now in main storage, the TESTRAN SVCs are inserted into the freshly loaded copy of the segment. The OPTEST count is again tested for zero or non-zero.

When the next TIA table in the OPTEST list has not been opened previously, a check is made to determine if it is now in main storage. If it is not, it is ignored and the OPTEST count is again tested for zero or non-zero. If, however, the TIA table is now in main storage the overlay 1 routine performs the normal test open function on the TIA table and then returns to check the OPTEST count for zero or non-zero again.

The Overlay 2 Routine (IEGTTRNX)

The overlay 2 routine (see Chart 36) is used whenever there are active traces in progress at the time an overlay occurs. The overlay 2 routine checks each table core to find the numbers of any inactive TIA tables. The trace table is searched for entries with matching TIA table numbers. Any such matching traces are stopped by removing their entries from the trace table. A trace stop record is written onto SYSTEST. If all active traces are stopped as the result of an overlay, the overlay stop bit is set in the trace stop flag byte of trace core. Exit from this routine is to the overlay supervisor via a RETURN macro-instruction. (Trace core is explained in detail in Appendix B.)

THE TRACE ROUTINES

Tracing operations within TESTRAN require the use of both setup and service routines. The trace interrupt routine is a setup routine, the other trace routines are service types and are discussed in the section entitled "The Service Routines."

The TRACE INTERRUPT Routine (IEGTTRNZ)

If it becomes necessary either to suspend or halt tracing during a trace operation, the trace interrupt routine (see Chart 38) is used. Depending on the event that triggered it, the routine initiates an abnormal-end-of-task (ABEND) sequence of events, transfers control to the tracer routine (via an XCTL macro-instruction), or returns control to the problem program (via the router routine). The following conditions cause the trace interrupt routine to be given control:

- Program check, which causes a transfer of control to the tracer routine if a user-supplied interrupt routine exists (SPIE macro-instruction). If no user-supplied interrupt routine exists, an ABEND results.
- Trace stop switch set on (stop flag byte in trace core), which writes an output message to indicate that tracing has been stopped and returns control to the problem program, providing the switch has been set by an overlay routine.
- Untraceable SVC (LINK, XCTL, or RETURN), which generates an output record and returns control to the problem program via the router routine.

Procedure for Decimal Instructions on Model 91

Trace interrupt routine modifications for handling decimal instructions encountered when operating on the Model 91 are described in this part under the section "Modifications of Trace Interrupt and Go Back Routines for the Model 91."

THE SERVICE ROUTINES

The service routines perform the test services specified by the TESTRAN macro-instructions. There are 16 service routines to accommodate the 23 TESTRAN macro-instructions: three TEST, three SET, two GO, five DUMP, and three TRACE. With the exception of the tracer routine, each of these routines is entered via a LINK macro-instruction issued by the router routine. The tracer routine is entered via an XCTL macro-instruction from the go back routine.

THE TEST ROUTINES

Except for the test open routine, which is a setup routine, the routines that correspond to the other TEST macro-instructions are service routines. These routines include: test on, test when, and test close.

The TEST ON Routine (IEGTTRNE)

The test on routine (see Chart 36) examines the value in the counter in the TIA entry to determine if the value is a multiple of the stipulated interval within the high and low limit specification. If the counter value meets the requirements, control returns to the router routine with the address in the VF pointer altered to show the address of the entry named as the operand of the TEST ON entry. If the requirements are not met, testing continues in an unaltered sequence.

Note: TESTRAN uses a register, called the varible field (VF) pointer, as a next entry indicator. The VF pointer is incremented as the router routine examines the TIA table entries. It points to the first byte of the next entry when the test services requested by the current entry have been executed. To transfer from the interpretation of one series of TIA table test request entries to another, the address of the first entry in the new series is substituted for the address of the next sequential entry in the VF pointer.

The pointer is required for normal sequential flow from one TIA entry to the next as well as when the normal flow is altered. Conditional entries such as TEST ON and TEST WHEN, and unconditional entries such as GO IN, GO OUT, and GO TO use the VF pointer to alter the sequence of interpretation.

The TEST WHEN Routine (IEGTTRNG)

The test when routine (see Chart 37) compares two operands specified in the entry. Depending on the result of the comparison, the routine either continues interpreting TIA entries or starts interpreting a new sequence beginning with the one specified in the test when entry. The user may specify one of six different "operators" upon which to base his comparison. These operators include:

- Equal to
- Less than
- Less than or equal to
- Not equal to
- Greater than
- Greater than or equal to

A comparison has a "true" result whenever the result conforms to the operator that was chosen. For example, if operand 1 has a lesser value than operand 2, and the operator chosen by the user is "less than", the result of the comparison is true. If, however, the same two operands were compared using "equal to" as the operator, the result of the comparison would be "false".

The two operands may be either flags or values in any one of several optional data modes (fixed point, floating point, hexadecimal, character, etc.). The test when routine has four comparison subroutines: one for fixed point values; one for floating point values; one for decimal values; and one that compares any of the hexadecimal-, character-, binary-, instruction-, or address-coded values.

The test when routine makes the comparison specified in its entry and, if the result is true, causes the address of the entry specified as the operand of the TEST WHEN entry to replace the address of the next sequential entry in the VF pointer. Thus, interpretation of the sequence of TIA table entries in which the TEST WHEN was located is terminated, and the interpretation of a new series is begun. A false result of the comparison causes a continuation of the interpretation of the current series of test requests.

The TEST CLOSE Routine (IEGTTRNH)

The test close routine (see Chart 39) closes the TIA table or tables that were opened by the test open routine when testing of the problem program was initiated. When the router routine encounters a TEST CLOSE entry in the TIA table, it links to

the test close routine which, in turn, reinserts the problem program instruction bytes that were displaced by the inserted SVC instructions. The routine retrieves the bytes from the reference table portion of table core and relinquishes the storage acquired for any table core associated with a closed TIA table.

Any trace operations started by entries in the closed TIA table are halted and a trace stop record is written onto SYSTEST. A test close record is also written onto SYSTEST. A bit is set on in the stop flags byte of trace core if all traces are stopped.

A device called the dummy go back, unique to the test close routine, is used to return control to the problem program. For entries other than a TEST CLOSE, a GO BACK entry (either explicit or implicit) eventually follows in the TIA table. In such cases, control returns to the problem program via the go back routine, the router routine, and the resident SVC routine.

After the test close routine is executed, the TIA table is no longer active, and no interpretation of an entry following the TEST CLOSE entry can be made. The VF pointer, therefore, is set to the address of a GO BACK entry in control core. This control core entry is labelled DUM and is a dummy GO BACK entry. Using the dummy go back as a device to pass control to the go back routine after the execution of a test close routine, control returns to the problem program, from the go back routine, in the normal manner.

THE GO ROUTINES

Although there are four GO macro-instructions, the TESTRAN interpreter uses only two routines to interpret them. The GO IN, GO OUT, and GO TO macro-instructions are interpreted by one routine; the GO BACK macro-instruction by the other. Routines for the GO macro-instruction entries unconditionally alter the sequence of TIA table entry interpretation when the router routine encounters their entries. The test on and test when routines alter the sequence of TIA table entry interpretation conditionally and are essentially additional go routines.

The GO IN/GO OUT/GO TO Routine (IEGTTRND)

Whenever the router routine encounters a GO IN, GO OUT, or GO TO entry in the TIA table, it passes control to the go routine (see Chart 40). The actual execution of the go routine, however, is different for each entry.

For a GO TO entry, the address, specified as the location of the TIA table entry to be interpreted next, is checked to determine if it is an entry in an active and open TIA table. If it is, the address is placed into the VF pointer register and control is returned to the router routine.

For a GO IN entry, the address of the next sequential entry that was in the VF pointer register is stored in the subroutine table for retrieval at the time a GO OUT macro-instruction entry is interpreted. (The subroutine table is a 9-byte table within control core. It can hold up to three such addresses.)

For a GO OUT entry, the address of the next entry to be interpreted is retrieved from the subroutine table and entered into the VF pointer register. (The address of the next entry for GO TO and GO IN entries is contained within the entry itself.)

Note: The subroutine table consists of three 3-byte entries. At each entry to the router routine (the execution of an SVC instruction inserted for a TEST AT entry in the TIA table), the subroutine table is zeroed. The go in routine fills the first vacant entry with the address of the next sequential TIA table entry following the GO IN itself. This address is used by the go out routine so that it can return to the next entry in the former sequence of entries.

Likewise, any subsequent GO IN entries enter the address of the otherwise next sequential entry into the subroutine table. If there are more than three entries into the table, the first entry is "lost". When a go out routine retrieves an address from the subroutine table, it takes the last one entered, and zeros that entry. Sequences of TIA table entries can be "nested" to a depth of three, the depth of the subroutine table.

The GO BACK Routine (IEGTTRNJ)

The go back routine (see Chart 41) is entered via a LINK macro-instruction from the router routine. This routine is entered whenever the router routine encounters an explicit or implicit GO BACK entry in the TIA table. A GO BACK entry is implied if the router encounters either an end-of-table marker entry, or a TEST AT entry signifying the beginning of a new

series of test requests. In either case, the series of entries being interpreted have come to an end.

With an explicit GO BACK entry, the user can specify the address (problem program address) to which he wishes control to be returned. The problem program is then reentered at a point other than the one from which the inserted SVC instruction caused it to be interrupted. If no address is specified in an explicit entry, the removed problem program instruction is reassembled and executed remotely. Control is then returned to the instruction following the inserted SVC via the router and resident SVC routines. The go back routine alters the address stored in the old PSW to make these returns possible.

If the trace mode switch has been set, the go back routine passes control to the TRACER routine instead of returning control to the problem program. If the user has supplied an SPIE macro-instruction to handle a program interruption, and that interruption occurs, the go back routine passes control to the user-supplied interruption handler routine. Otherwise, if a program interruption occurs, the go back routine initiates an ABEND.

Procedure for Decimal Instructions on Model 91

Go back routine modifications for handling decimal instructions encountered when operating on the Model 91 are described in this part under the section "Modifications of Trace Interrupt and Go Back Routines for the Model 91."

THE SET ROUTINES

Each SET macro-instruction has its own service routine. The SET routines receive control from the router routine via a LINK macro-instruction. When their function is completed, control returns to the router routine.

The SET FLAG Routine (IEGTTRNP)

The first operand in the TIA table entry for the SET FLAG being interpreted is checked to ensure that it is in the flag table of an open and active TIA table. The second operand may also be a flag. If so, it is also checked to determine whether or not it is in an active flag table. If the first operand is an active flag and the second operand is either an active flag or a specific value, the first operand is set to the value of the second. If the second operand is a valid storage address, the first operand is set to the value of the contents of the second operand. (See Chart 42.)

The SET COUNTER Routine (IEGTTRNN)

The first operand in the TIA table entry for the SET COUNTER being interpreted is checked to ensure that it is in the counter table of an active and open TIA table. Likewise, the second operand, if it is a counter, is checked to determine whether or not it is in an active counter table. If the first operand is an active counter and the second operand is either an active counter or a specific value, the first operand is set to the value of the second. If the second operand is a valid storage address, the first operand is set to the value of the contents of the second operand. (See Chart 42.)

The SET VARIBLE Routine (IEGTTRNR)

The first operand in the TIA table entry for the SET VARIBLE being interpreted is checked to ensure that it is within the user's task; i.e., either a register or a field in main storage. The second operand is checked to ensure that it is either a valid storage address, an active counter, or a literal. If the checks indicate that both operands are valid, the first operand is set to equal the register, literal, or counter value of the second operand. If the second operand is a valid storage address, the first operand is set to the value of the contents of the second operand. (See Chart 39.)

THE TRACE ROUTINES

There are three trace routines to interpret the four TRACE macro-instructions. Two routines, trace start and tracer, are executed for the TRACE CALL, TRACE FLOW, and TRACE REFER entries; the other, for the TRACE STOP entry. The trace service routines are supported by the trace setup routine (trace interrupt).

Whenever the go back routine is entered, it interrogates the trace mode switch (a flag bit in control core). The trace mode switch is set on by the trace start routine, when the first trace is started, to indicate that any subsequent trace is not the first and that a table called "trace core" already exists. If the trace mode switch is off, the storage required for trace core and the table contained within it (trace table) must be acquired and the tables initialized. The address of the trace table (and trace core, since the trace table is the first portion of trace

core) is entered into a field of control core. A record-pair, containing a prologue record that includes a type byte to indicate what type of trace has been started, is written as output from the execution of the TRACE START routine. The additional data describing the trace, addresses for start and stop limits of the trace, is written into the data record. (Detailed descriptions of trace core and trace table are in Appendix B.)

To perform a trace operation, TESTRAN routines copy a problem program instruction from its location in the program to a remote location (low end of main storage), examine it, then remotely execute it at that location. The exact procedure depends on the problem program instruction and the type of trace being made.

The low order 48 bytes of main storage are used temporarily by the tracer routine, their original contents are saved and later restored. The contents of the general registers, as they appeared in the problem program, are restored before the instruction is executed and then preserved after it is executed.

On each entry to the tracer routine, the stop flags byte in trace core is examined to ensure that no event has stopped all traces since the last entry. Except for program check interruptions that are enabled by the problem program, all interruptions are disabled. Six bytes of problem program, beginning with the next instruction to be executed, are moved to location zero in main storage. The instruction is then examined and the course of action to be taken is determined.

Should the instruction type cause a branch in the problem program, it is first determined whether or not the branch should occur. If the branch would occur during a normal untraced operation it must be determined whether or not a TRACE FLOW or TRACE CALL is in effect and whether or not this branch requires an output record to be generated. If a record is necessary, it is written onto SYSTEST to indicate the branch. The pointer used to indicate the problem program instruction to be examined next is altered so that it now contains the "branch to" address.

If the instruction examined is a store type and a TRACE REFER is in progress, the routine changes the storage protection key to that of the problem program and writes a "before" record of the area onto SYSTEST. It then executes the instruction and restores the storage protection key to zero. If the conditions required for a "before" record to be written were met, an "after" record must also be written.

If the examined instruction is an execute, it is first determined that the operand instruction is not also an execute, since it is invalid to execute an execute. If the operand instruction is a valid type it is moved (six bytes) to location zero and the determination of its type is begun just as though it were an "in line" instruction. The next-instruction-to-be-examined-pointer is kept pointing at the instruction that follows the execute.

For certain traceable SVC instruction types encountered, a trace stop record is written and control is returned to the router routine. If the generation of a trace flow output record is required, it is written.

With privileged instructions, control passes to the trace interrupt routine via an XCTL macro-instruction.

For all instructions other than those mentioned above, the instruction is moved to location zero, and executed. Interruptions are enabled, and the process starts again with the next problem program instruction.

The TRACE START Routine (IEGTTRNL)

The trace start routine (see Chart 37) initiates trace operations within TESTRAN by writing a trace start record and initializing trace core and the trace table. For the starting of subsequent traces, it adds an entry to the trace table for each new trace.

Note: Since the trace table is limited to ten entries, the number of concurrent traces is also limited to ten. Should the number of traces exceed ten, the last trace started is "lost". (The trace table is sorted by trace type and the last entry is flagged. An entry for a trace after the tenth one replaces the previous last entry.)

Following the execution of the trace start routine, control returns to the router routine. If the router determines that the next entry in the TIA table represents another test action or test control macro-instruction to be acted upon, it proceeds to do so, ignoring the fact that the trace mode switch is on.

If, however, the next entry is a GO BACK (either explicit or implicit) and the go back routine finds that the trace mode switch is set on, instead of returning control to the problem program, the go back

routine passes control to the tracer routine so that it may monitor each problem program instruction. (Even if the entries directly after the TRACE entries are other than GO BACK entries, this condition will eventually occur.)

The TRACER Routine (IEGTTRNT)

The tracer routine does not interpret TIA table entries directly, but rather, with the trace start routine provides the tracing services. (See Chart 43.)

When the router routine passes control to the go back routine at the completion of a sequence of test requests, the go back routine tests the status of the trace mode switch. If the trace mode switch is set, the go back routine passes control to the tracer routine, which examines each problem program instruction and executes it remotely.

If a TRACE CALL is being executed, each CALL macro-instruction executed within the area specified by the TRACE CALL entry in the TIA table causes an output record pair to be generated and written onto SYSTEST.

If a TRACE FLOW is being executed, each branch to or from the areas specified in the TRACE FLOW entry in the TIA table causes an output record pair to be generated and written onto SYSTEST.

If a TRACE REFER is being executed, each instruction that alters the contents of any area specified in the TRACE REFER entry in the TIA table causes an output record pair to be written onto SYSTEST.

Should the tracer routine encounter a TESTRAN SVC instruction while it is executing the problem program, control returns to the router routine.

Whenever a privileged instruction is encountered, whenever a program check occurs, or whenever an untraceable SVC (LINK, XCTL, or RETURN) is encountered in the problem program, control passes from the tracer routine to the trace interrupt routine.

A machine check interruption that occurs while the tracer routine is in operation results in an ABEND.

Note: While the tracer routine is executing the problem program, interruptions are enabled and then disabled after each problem program instruction is executed. SVC called routines such as the overlay supervisor are not traced.

The TRACE STOP Routine (IEGTTRNM)

The trace stop routine (see Chart 37) is entered when the router routine encounters a TRACE STOP entry in the TIA table. If the entry has no specific traces named as operands, all traces are stopped and their entries are removed from the trace table. If, however, specific traces are named as operands, each trace so named is stopped and its entry is removed from the trace table. A bit in the stop flags byte of trace core is set on if all traces are stopped by a TRACE STOP entry interpretation.

Each entry in the trace table contains data to identify its type, its output-record priority codes, its TIA table and macro-instruction numbers that correspond to this trace, and the start and stop limits of the trace.

THE DUMP ROUTINES

Five dump routines are used to interpret the six DUMP macro-instructions. The DUMP DATA and DUMP CHANGES macro-instructions are interpreted by one routine; the other four by their respective dump routines. All of the routines are entered from the router routine via a LINK macro-instruction.

The DUMP DATA Routine (IEGTTRNA)

Whenever the router routine encounters a DUMP DATA or DUMP CHANGES entry in the TIA table, it links to the dump data routine (see Chart 40). The dump data routine determines whether these addresses are within the physical size limitations of the computing system, whether the start address is lower than the stop address, and whether the area requested is more than 65,535 bytes long. If no violations are discovered, the contents of the area specified is written onto SYSTEST.

Except for a unique type byte in the prologue, the same output is generated for both a DUMP DATA entry and a DUMP CHANGES entry if both are concerned with the same area of storage.

The DUMP COMMENT Routine (IEGTTRNB)

Whenever the router routine ineteprets a DUMP COMMENT entry in the TIA table, it links to the dump comment routine (see Chart 40). This routine generates a prologue record that contains a DUMP COMMENT type byte and the identification numbers of both the TIA table being interpreted and the macro-instruction whose TIA entry caused this entry to the dump comment routine. The record-pair is written onto SYSTEST, with the data record of the pair consisting of a dummy record. The actual comment itself is not written onto the data set. (The TESTRAN retrieves the comment from the TIA table at edit or edit time.)

The DUMP MAP Routine (IEGTTRNK)

The router routine links to the dump map routine (see Chart 40) whenever it encounters a DUMP MAP entry in the TIA table. The dump map routine generates an output record-pair that consists of a prologue record and a data record. The data record contains a series of subrecords acquired from system tables, one for each field of storage associated with the current task. The number of subrecords is indicated in the data record. Each subrecord contains information as to whether that area is a program or a data area, the name of the program to which the area is assigned, and the beginning address and length of the area. After the record-pair is written onto SYSTEST, control returns to the router routine.

The DUMP PANEL Routine (IEGTTRNC)

Whenever the router routine encounters a DUMP PANEL entry in the TIA table, it links to the dump panel routine (see Chart 40). This routine retrieves the contents (at the time of the inserted-SVC-caused interruption) of the 16 general registers from the register save area. It also gets the contents of the floating-point registers (if present) directly from the registers, and the contents of the old PSW. An indentifying prologue record and a data record that contains the data items just described are written onto SYSTEST. When the TESTRAN editor processes the record at edit time, it selects the required registers.

The DUMP TABLE Routine (IEGTTRNF)

The dump table routine (see Chart 31) is linked to by the router routine whenever it encounters a DUMP TABLE entry in the TIA table. The DUMP TABLE entry specifies either the TCB, DCB, or DEB. If a TCB is specified, the task control block of the requesting task is written out as the data record portion of a record-pair. If a DCB is specified, the named data control block is written out as the data record portion of the record-pair. If a DEB is specified, the named DCB is examined and the address of the data extent block is obtained from the field in the DCB. The contents of the DEB are written as the data record. When the function of this routine is completed, control returns to the router routine.

MODIFICATIONS OF TRACE INTERRUPT AND GO BACK ROUTINES FOR THE MODEL 91

In the simulation of problem program instructions while operating in either the trace mode or the go-back mode of TESTRAN, a program interruption occurs on the Model 91 whenever the TESTRAN interpreter encounters a decimal instruction of the problem program. The TESTRAN Trace and Go-back routines have been modified to permit the TESTRAN interpreter to utilize the facilities of the Decimal Simulator (IEAXDS00) routine via the PFLIH routine.

When either the TESTRAN Trace Interrupt routine (IEGTTRNZ) or the TESTRAN Go-back routine (IEGTTRNJ) receives a program check interruption caused by an invalid operation code, a 'routine-to-TESTRAN' PSW and an error return address are established in the TESTRAN Control Core. (The TESTRAN Control Core contains the PSW at location 448 and the error return address at location 444.) The high-order bit (called the 'return-to-TESTRAN' flag) in the TESTRAN Control Core Pointer (i.e., the TCBTRN field) in the TCB is set to 1 to indicate that the TESTRAN program is entering the PFLIH routine. TESTRAN saves the contents of the general registers 9 through 8 beginning at location 456 in the control core. Control is then given to the PFLIH routine.

If the decimal instruction that TESTRAN gives to the PFLIH routine, and which subsequently is given to the Decimal Simulator, is simulated successfully, the Decimal Simulator routine returns control to TESTRAN. Return is achieved by locating the 'return-to-TESTRAN' PSW and then using the LPSW instruction.

If the decimal instruction is not simulated successfully, the Decimal Simulator routine returns control to the PFLIH routine, which checks the 'return-to-TESTRAN' flag to ensure that the entrance to the PFLIH routine was from TESTRAN. The address of the TESTRAN error-handling routine is moved into the 'return-to-TESTRAN' PSW, and control is given to the error routine. The TESTRAN interpreter recognizes the type of error that was detected by the Decimal Simulator routine by testing

the program old PSW. The functions of the TESTRAN error-handling routine are unchanged.

INTERPRETER LOGIC FLOW EXAMPLE

Figure 3 shows the flow of control through a simplified problem program/TIA table execution. The problem program/TIA load module includes CSECTs for both the problem program and the TIA table.

The load module uses SVC instructions and RETURN macro-instructions to communicate with the TESTRAN interpreter's test open routine and router routine (setup routines). The router routine communicates with the TESTRAN interpreter's dump, go back, tracer, and trace stop routines (service routines) via LINK and RETURN macro-instructions. An XCTL macro-instruction is used for communication between the go back and tracer routines.

After program fetch enters the load module into main storage and the contents supervisor determines that the load module has test attributes, control passes to the TSO TEST save routine via an SVC 61 instruction. The TSO TEST save routine will XCTL to the TESTRAN interpreter's save routine if the TCB under which the SVC 61 instruction was issued does not indicate a TSO task.

The save routine retains the address data used by program fetch when it acquired the load module from its external residence and then returns control to the contents supervisor. The contents supervisor, in turn, passes control to the entry point specified for the load module; i.e., the TEST OPEN entry in the TIA table.

This entry, executable as an SVC 49, causes an interruption and transition from problem to supervisor states. The test open routine obtains storage for its tables and initializes them. It inserts an SVC 38 instruction into the problem program at each point specified in a TEST AT entry in the TIA table.

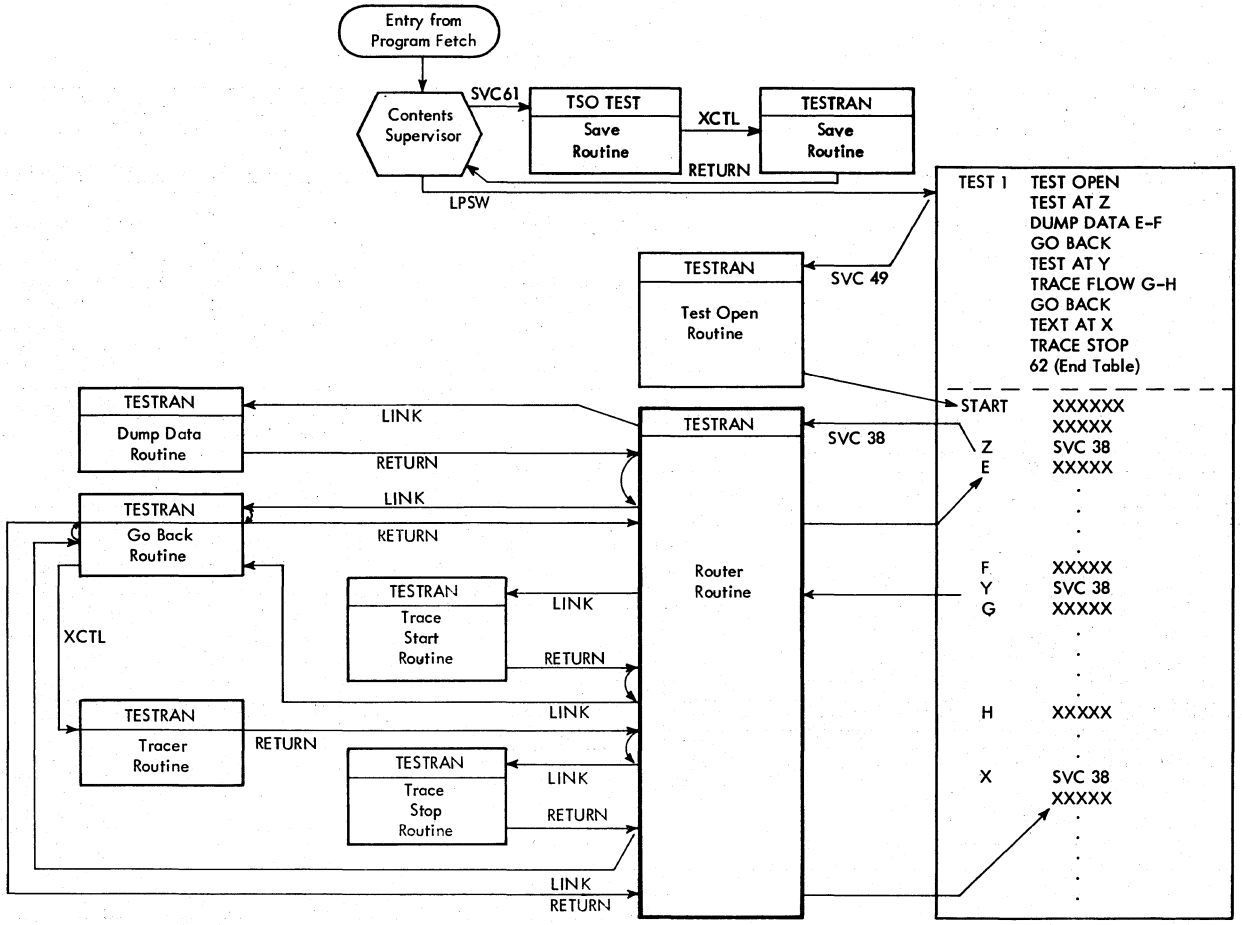


Figure 3. TESTRAN Interpreter Logic Flow Example

The test open routine copies onto the output data set the assembly time symbol tables, and external symbol dictionary (ESD) data acquired from their external residence using the address data retained by the save routine. It also writes a composite ESD (CESD) record, a relocation table record, and records consisting of copies of the TIA and reference tables. In addition, it writes a test open record for the editor, indicating that test open has occurred. It then loads the TESTRAN interpreter's router routine into main storage and passes control to the problem program at the point specified in the TEST OPEN macro-instruction; i.e., the instruction labeled START.

Control is in the problem state and remains in this state and within the problem program until an inserted SVC 38 instruction is encountered. Control then goes to the supervisor state and through the resident SVC routine to the router routine, which determines the SVC 38 instruction that caused this interruption. This routine also determines which TESTRAN test request is coded into the TIA table immediately after the TEST AT entry that caused that SVC to be inserted into the problem program.

In Figure 3, location Z in the problem program CSECT is specified as the point at which the tests are to begin. The next entry in the TIA table specifies DUMP DATA E through F. The router routine examines the entry and passes control, via a LINK macro-instruction, to the DUMP DATA routine, which operates in the supervisor state. After the contents of the main storage area between instructions E and F are written onto the SYSTEST, control returns to the router routine.

The router routine determines that the next TIA entry is a GO BACK entry and passes control to the go back routine via a LINK macro-instruction. The go back routine executes the problem program instruc-

tion displaced when the SVC 38 was inserted into the problem program. It then returns control to the problem program at instruction E. Control is again in the problem state. It remains in this state through instruction F.

When instruction Y is executed, another SVC 38 causes an interruption and a transition from problem to supervisor state occurs. As a result of the interruption, the router routine receives control. It then locates the TIA table entry that follows the proper corresponding TEST AT entry; i.e., TEST AT Y.

Since the next TIA entry is a TRACE FLOW G through H, the router routine passes control to the trace start routine to initiate the trace. This routine establishes the tables and entries necessary for trace operations. It also generates an output record to indicate that the trace has been initiated and sets a switch to show that a trace is in progress. When this operation is completed, control returns to the router routine. The router routine examines the next sequential entry in the TIA table to determine the next routine to get control; i.e., a GO BACK entry. Control, therefore, passes to the go back routine, which reassembles the displaced problem program instruction and then examines the switch that indicates whether or not a trace is in progress. (It examines this switch each time it receives control.) Since there is a trace in progress and the switch is set, the go back routine does not return control to the problem program, but rather passes it to the tracer routine.

The tracer routine operates within supervisor state but executes the problem program. It examines the problem program instruction-by-instruction and executes it remotely.

Each instruction is "lifted" from the problem program sequentially, beginning with the one displaced by the SVC, and examined to determine whether it is a program-transfer type instruction or an execute that causes a program-transfer type to be executed. If it is such an instruction within the specified limits of the trace, an output record is generated to describe the action.

In Figure 3, control remains with the tracer routine until the routine encounters the SVC 38 at location X. As the result of encountering the SVC, the router routine receives control and it again finds the proper entry in the TIA table to be interpreted; i.e., the TRACE STOP entry. As a result of the execution of the trace stop routine, all traces in progress are stopped. In this case, since the TRACE FLOW is the only trace in operation it is the only trace that is stopped. The trace stop routine removes the trace table entry that identifies the trace. An output message, which indicates that the trace was stopped, is written and control returns to the router routine.

The router routine again determines the next entry in the TIA table; i.e., an end-table entry, (which is an implied GO BACK). The router routine passes control to the go back routine. The go back routine reassembles the displaced problem program instruction and executes it. Control then passes via the resident SVC routine to the problem program, which continues to execute without further TESTRAN steps.

The TESTRAN editor is a post processor that functions after the execution of the problem program. Operating completely within problem state, it converts test output data generated by the TESTRAN interpreter into a printable output which the user can employ to debug his problem program. The editor is a separate job or job step and is scheduled and controlled in the same way as any problem program or processor.

Thirty-eight routines make up the TESTRAN editor. These routines are divided into the following:

- Initialization routines.
- Internal routines.
- Output routines.

All the TESTRAN editor's routines reside on the link library (SYS1.LINKLIB) partitioned data set. Control is passed among the routines via LINK, XCTL, and RETURN macro-instructions and to the routines of the root module via direct branches.

The overall program structure of the TESTRAN editor is shown in Figure 4. The editor's routines are used in a variety of patterns which are determined by the sequence of records read from the input data set. The input data set processed by the editor is the TESTRAN interpreter's output data set (SYSTEST).

In addition to test output data generated by the TESTRAN interpreter's service and setup routines, the interpreter's test open routine writes records that contain copies of the TIA and reference tables, the assembler symbol table (SYM), and external symbol dictionary (ESD), the composite external symbol dictionary (CESD), and the control section relocation table. If the problem program is an overlay program, the test output data also includes map change records. These records, when available, are used by the TESTRAN editor to prepare symbolically labeled output records for the user. Although the editor can function without most of these input data records, it cannot function without a copy of the TESTRAN interpreter's TIA table.

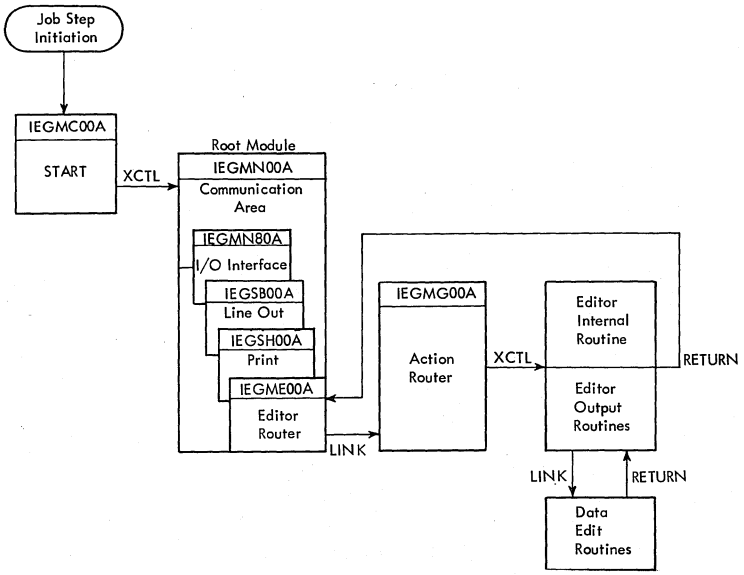


Figure 4. TESTRAN Editor Organization

OPERATION OF THE TESTRAN EDITOR

The TESTRAN editor is a separately scheduled job step that must be executed after the execution of a problem program which has been under test with TESTRAN. For the editor to operate, a TIA table for testing the problem program must have been assembled and linkage edited into a load module with the problem program, following which the program must have been executed. The interpreter setup and service routines must have generated some test output data and written it onto a secondary storage data set whose ddname is SYSTEST. This data set must be made available as input to the editor at its execution time.

The editor provides the user with a printed output to use in debugging the problem program. Storage dumps are identified with loaded and assembled addresses, and symbolic labels are applied to data items they name. Dumps of program areas of storage can be printed with the assembler language mnemonics inserted, floating point data items can be printed in decimal form, etc.

The TESTRAN editor builds and uses several tables which are essential to its operation. The various tables serve to establish the environment of the problem

program as it existed when each test output data record was generated.

EDITOR TABLES

The editor alters and updates its tables throughout its execution. This continuing process of keeping the tables current, relative to the problem program's execution, is the job of the internal routines. The tables include the table dictionary, the map, the action table, the dump change table, the symbol table, and the reference table. Diagrams and field-by-field breakdowns of these tables are contained in Appendix C.

TABLE DICTIONARY

The table dictionary contains four 8-byte entries. These entries give the starting address and the current length of the map, the reference table, the action table list, and the dump change list.

MAP

The editor's map contains a series of 24-byte entries. Each entry is updated continuously, throughout execution of the editor, to show the identification, the status, and the significant addresses of a defined section of the problem program. Defined sections include CSECTs, DSECTs, unnamed or private code CSECTs, and blank common control sections. The number of entries in the map depends on the number of program sections that are loaded. The maximum size of the map is determined during initialization, and is one of the two possible sizes indicated in the listing.

ACTION TABLE LIST

The action table list contains a series of 10 byte entries. The entries give the secondary storage address and identification needed by the editor's routines to locate the action table and a specific entry within the action table. The number of entries in this table is dependent on the number of test open routine executions during the execution of the problem program. Maximum size for this table is indicated in the listing.

ACTION TABLE

The action table is a version of the TESTRAN interpreter's TIA table which has been altered by an editor internal routine to eliminate data not needed and to fix the length of some variable fields. The original interpreter TIA table has up to 23 entry types, whereas this altered version retains only 9 entry types. If it exceeds the size of the buffer used, the altered TIA table (action table) is written on secondary storage as it is altered. The action table list permits the retrieval of the action table entries. A fixed length buffer in main storage is used to form the secondary storage records. This buffer is also used by the symbol table initializer routine.

DUMP CHANGE LIST

The dump change list is a series of 5-byte entries used to point to the secondary storage residence of a dump change table. Each entry gives the TIA table number and the number of the DUMP CHANGES macro-instruction entry within that TIA table, plus the address on secondary storage of the dump change table. The number of entries in this table (list) depends on the number of DUMP CHANGES entries interpreted at problem program execution time. The maximum size of the list is determined at initialization and is one of two possible sizes as indicated in the listing.

DUMP CHANGE TABLE

The dump change table contains a series of 9-byte entries. This table is maintained on secondary storage. It indicates the loaded address (at problem program execution time), the length, and the secondary storage location of the specified data. This data is the last copy of the area of storage referred to in a dump changes TIA entry. It is the copy of the storage area with which the editor compares the copy in a dump changes record it is processing.

SYMBOL TABLE

The editor's symbol table is an altered version of the assembler's symbol table which was passed to the editor by the interpreter test open routine. The symbol

table is stored on secondary storage in records of a fixed length (size of symbol table buffer) for each section definition. An entry is made in the editor's map to identify the first record of each symbol table. Each record of variable length symbol table entries is given a heading which indicates the last (highest) offset from the origin, into the defined section, of the current record. The heading also indicates whether that record is the last record of a table. This heading data speeds the search for symbol information needed by the editor routines.

REFERENCE TABLE

The TESTRAN editor's reference table contains a series of six byte entries. It is an altered version of the TESTRAN interpreter reference table. It contains the displaced instruction bytes and their locations in the problem program plus the segment number of the associated TIA table. It is used by the editor in showing what instruction was displaced, in dump type output records containing inserted SVCs.

The number of entries in this table is dependent on the number of TEST AT entries in the interpreter TIA table. The maximum number of entries is determined at initialization and is one of two possible limits indicated in the listing.

INITIALIZATION ROUTINES

The Start Routine (IEGMC00A)

At edit time, control passes initially to the start routine. (See Chart 50.) This routine gets main storage for input record buffering. It tailors the table sizes to the amount of storage available. The start routine processes parameters from the job control EXEC statement (PARM=). It also opens the three data sets used by the editor: the input data set (SYSTEST), the intermediate data set (SYSUT1), and the output data set (SYSPRINT). SYSTEST is the output data set from the interpreter, generated at problem program execution time. SYSUT1 is the editor's scratch or working data set maintained on secondary storage. SYSPRINT is the data set upon which the editor's output print data is written.

The start routine passes control to a LOAD macro-instruction routine to call the editor's root module (IEGMN00A) into main storage. The tables and buffers used by the editor are initialized by the start routine after it has acquired the necessary storage. The area of main storage occupied by the start routine is relinquished when it passes control to the root module via an XCTL macro-instruction.

The Root Module (IEGMN00A)

The root module is divided into a communications area, an I/O interface, and three discrete routines; the print routine, the line out routine, and the editor router routine.

The module is loaded into main storage by the start routine and remains resident in main storage throughout the editor's execution. The root module is the only editor module that is maintained thus. Control is passed to the module's routines from other editor routines via a branch list.

Print Routine (IEGSH00A): The print routine writes the editor's output records onto SYSPRINT. This routine is device independent. It may or may not actually print the output records. When control passes to the print routine, the current 120-character buffer is written onto SYSPRINT and the line count total is incremented. To write the record onto SYSPRINT, the print routine calls upon the I/O interface routine to establish communication between it and data management's BSAM which performs the writing function.

When the maximum line count per page is reached, a page eject is initiated and a new header record written before the next output record can be written. Once an output record is written, its buffer is cleared.

The print routine checks the output page count. If the established maximum page count is reached, a special exit from the print routine is taken.

Line Out Routine (IEGSB00A): The line out routine builds the print lines written onto SYSPRINT by the print routine. Data to be printed is indicated to this routine through designated general registers and parameters that are contained within the communications area. Information concerning the source of the data, the length of the data, and the number of blanks to follow the data are all passed to this routine.

A pointer in the communications area, indicating the current position in the current buffer into which data can be

added, is used to indicate where the data fits into the buffer. The line out routine updates this same pointer and checks for attempted line overflows, passing control back to the calling routine through an overflow exit without entering anything in the line if such an overflow attempt occurs.

Editor Router Routine (IEGME00A): The editor router routine (see Chart 50) controls the reading of 21 input record types from SYSTEST and makes an initial analysis of the record type. Depending on that analysis, this routine links to the action router routine for records that require the selection of a routine to process them, or back to the routine that processed the previous record for a continuation type input record.

I/O Interface (IEGMN80A): All reading and writing of input, output, and secondary storage data within the TESTRAN editor uses the I/O interface portion of the root module. This portion contains the DCBs, the DECBs, and the buffering routines used by the editor in its interface with BSAM.

Communications Area (IEGMN00A): The communications area of the root module contains the constants needed by the editor's other routines to maintain contact with one another. It contains the constants and common address parameters used by the other routines and a branch list to allow control to be passed from a routine to one within the root module.

The Editor Message Routine (IEGSF00A)

All error and informational-type messages issued by the editor are written under control of the editor message routine. (See Chart 51.) The message text is stored within the calling routine. Whenever an editor routine requires a message to be written onto SYSPRINT, it links to this routine. The editor message routine, in turn, calls the line out routine to move the selected message to the current print buffer, and the print routine to write the message onto SYSPRINT. At the conclusion of its functions, the editor message routine returns control to the editor routine that issued the message request.

INTERNAL ROUTINES

The TESTRAN editor's internal routines process the input record types that provide information concerning the problem program whose test output data is being edited. This information is maintained by the editor in the form of storage maps and tables. Each time the problem program's storage configuration changes due to an overlay, the TESTRAN interpreter writes a map change record which is essentially a copy of the overlay segment table.

Each map change record is applied to the storage map by the internal routines to reflect the current (relative to problem program execution) status of storage. Any output records that follow can then be applied to the proper program section and any symbolic data available can be assigned to the proper data. Other internal routines update tables used by the editor, generate editor altered versions of interpreter tables, process symbol table records, and provide end-of-run processing when needed.

The Reference Table Routine (IEGRK00A)

The reference table routine (see Chart 52) processes reference table type records read from SYSTEST. It refers to the prologue record or the editor's map to determine the overlay segment number of the TIA table associated with this reference table, and adds this information to the data in the reference table entry from the interpreter. The address of the TEST AT macroinstruction entry in the interpreter's TIA table is deleted from the reference table entry and the resulting altered form of the entry is stored in the editor's reference table. When it completes the processing of the reference table record, this routine returns control to the editor router routine.

The Action Table Routine (IEGRL00A)

The action table routine (see Chart 52) is linked to by the action router routine when the router routine determines that the current input record is a TIA table type. The routine eliminates all TIA table entry types except for the test open, dump, and trace types. These entries are altered to eliminate non-useful data An editor action table is formed from them. As the fixed length buffer used to form the table is filled it is written onto secondary storage. All the action table entries are presented in Appendix C.

The Invalid Record Routine (IEGPE00A)

The invalid record routine (see Chart 53) keeps an invalid record count that is updated each time an unreadable or invalid-type record is encountered as input to the TESTRAN editor. If the invalid record count reaches three, the input is considered to be uneditable and the editor run is terminated. Control passes to the editor message routine, so that a message, indicating that the count of invalid records has exceeded the limit, can be issued.

If the count has not reached three, an invalid record message is issued and control is returned to the editor router routine. Should the count reach the limit, after issuing both the invalid record and excessive invalid records messages, the invalid record routine passes control to the end-of-run routine to terminate the job.

The End-of-Run Routine (IEGPK00A)

The end-of-run routine (see Chart 53) is entered via an XCTL macro-instruction from the invalid record routine. An actual end-of-data-set condition from SYSTEST, exceeding the maximum-pages-of-output limit, an uncorrectable I/O error, or an invalid CSECT relocation table record also causes control to be passed to this routine. The routine writes (on SYSPRINT) an end-of-job message, a maximum pages message or an I/O error message (when applicable), and a count of statements processed. It also closes the data sets opened by the editor and returns control to the supervisor.

The Relocation Table Routine (IEGRE00A)

The relocation table routine (see Chart 52) processes CSECT relocation table records written by the TESTRAN interpreter. Since the editor's map is kept sorted on the loaded address field, the order of entries within the map does not correspond to the sequence in which the entries were made. Each entry in the map, therefore, is given a sequence number.

When a CSECT relocation table record is processed, the relocation table routine applies each relocation factor to the map in sequence number order (i.e., from the lowest numbered CSECT to the highest numbered one). Following the application of the relocation factors, this routine searches the program storage area for defined sections which occupy storage identified in the map as belonging to some other defined section. Such a situation develops when using LOAD, DELETE, LINK, and XCTL macro-instructions but not through an overlay. In such circumstances, the entry in the map that details the section that has been "stored over" is purged from the map. The routine again sorts the map on the loaded address field. It then returns control to the editor router routine.

If the relocation factor cannot be correlated with the map, a message is issued and the editor run is terminated.

The Map Change Routine (IEGRC00A)

The map change routine (see Chart 54) processes the map change record from the interpreter and applies the data in the record to the TESTRAN editor's map. The storage map contains information that concerns the storage in use by the problem program at the time testing was taking place. The map change record is written by the interpreter because an overlay of the problem program has altered the contents of storage.

CSECTs and their associated TIA tables, which were active and may have been generating test output data previous to the overlay, may have been overlaid and no longer be active. For the editor to apply the proper set of symbolic labels to the test output data it is processing, it must keep its storage map current, relative to that data. The map change routine alters the map to keep it current, reflecting the changes from active to inactive (or vice versa) of all CSECTs so changed. At its completion, it returns control to the editor router routine.

The CESD Map Routine (IEGRA00A)

When the action router routine (see Chart 53) determines that the input record from SYSTEST is a CESD record, it passes control to the CESD map routine via an XCTL macro-instruction. If an editor map exists, the addresses assigned by the linkage editor and contained in the CESD record are inserted in it, and sequence numbers are assigned to the CSECTs in the map. If no map exists when a CESD record is read, this routine creates one from the available information. At the conclusion of its operations, this routine returns control to the editor router routine.

The Symbol Table Processing Routines

Five routines are used by the TESTRAN editor to process symbol table (SYM) records. The routines process both the assembler SYM records and the assembler's external symbol dictionary (ESD) records. The output from these routines is written onto secondary storage (SYSUT1). This secondary storage data is in the form of an altered symbol table arranged to expedite retrieval by the other editor routines.

Symbol Table Base Routine (IEGRF00A): The symbol table base routine (see Charts 55 and 56) receives control, via an XCTL macro-instruction, from the action router routine when it finds that the input record to be processed is a symbol table record. The routine establishes addressability of the communication area for the SYM routines. It contains subroutines necessary to get the variable length SYM records from the input area and to build and search the editor map.

This routine remains resident in main storage throughout the SYM processing with control being passed to its subroutines from each of the other SYM routines.

Symbol Table Initializer Routine (IEGNS00A): The symbol table initializer routine (see Charts 55 and 56) receives control from the symbol table base routine via a LINK macro-instruction. Initial entry to this routine causes the contents of the action table buffer to be saved by writing it onto secondary storage. This frees the buffer for use during the execution of the symbol table initializer routine. The routine causes the editor map base to be established. Subsequent entries to this routine are via an XCTL macro-instruction from the symbol table last pass routine at the conclusion of SYM processing for each assembly of the load module. If more data (SYM or ESD records) is to be processed the symbol table initializer routine operates and then passes control, via an XCTL macro-instruction, to the symbol table first pass routine. Otherwise, the action table buffer is restored, and control is returned to the editor router routine.

Symbol Table First Pass Routine (IEGNY00A): The symbol table first pass routine (see Charts 55 and 56) is entered from the symbol table initializer routine via an XCTL macro-instruction. It processes the variable length input SYM records into fixed length records, sorts these records, and writes them onto SYSUT1.

This routine exits, via an XCTL macro-instruction, to the symbol table ESD routine when the SYM records (in the input) end and are followed by the ESD records.

Symbol Table Last Pass Routine (IEGSP00A): The symbol table last pass routine (see Chart 57) is entered from the symbol table base routine at the conclusion of processing SYM records from each assembly. It gets the fixed length records formed by the symbol table first pass routine from their secondary storage residence and processes them into variable length records broken down by CSECT. The new strings of symbol table records formed by the preceding process are again written onto secondary storage (SYSUT1) and the I/O address of the symbol table data for each CSECT is placed in the editor map entry for that CSECT.

The symbol table last pass routine exits, via an XCTL macro-instruction to the symbol table initializer routine to process any following SYM records.

The Symbol Table ESD Routine (IEGRG00A): The symbol table ESD routine (see Chart 58) processes the ESD records read in as part of the SYM records. If the record is a section definition (CSECT, DSECT, etc.), the symbolic name is inserted into the editor map. The routine is terminated by the symbol table base routine at the end of each assembly's SYM records so that the symbol table base routine can get the next input record if it is a SYM record.

OUTPUT ROUTINES

The output routines produce output records from the data supplied by the TESTRAN interpreter. They are supported by the initialization and internal routines. The output routines prepare symbolically labeled output data in the form of storage dumps, and traces for the user to employ in debugging his problem programs. The initial determination that a record read in by the editor is other than a continuation type is made by the editor router routine. When the editor router routine detects such a record, it passes control to the action router routine so that it may determine which routine is needed to process that record.

The Interpreter Message Routine (IEGPI00A)

The interpreter message routine (see Chart 53) receives control from the action router routine for either of the two interpreter message record types. The records contain the code used by the interpreter

message routine to select the proper message from the table of messages contained within the routine itself. Depending on which of the record types the routine is processing, the executed statements line may or may not be printed. To identify the message as a TESTRAN interpreter message, this routine prefixes the characters "*** IEGI" to the head of the message.

If the message includes an address as a suffix, this routine links to the address analyzer routine to allow display of the address in symbolic form. It uses the line out and print routines of the root module.

This routine exits, via a RETURN macro-instruction, to the editor router routine.

The Action Router Routine (IEGMG00A)

If the editor router routine determines that the record from SYSTEST is to be processed, it passes control to the action router routine (see Chart 54). The action router routine further analyzes the record to determine whether it requires an output routine to process it, and if so whether its output selection code is one which is to be processed. If the record does not have one of the selected output selection codes, it is ignored. Control remains with the action router until any continuation records for the non-selected record have been read, following which control is returned to the editor router routine.

When the routine processes an output record, it formats the "AT LOCATION" (where applicable), the "EXECUTED STATEMENTS", and the "CURRENT ACTION" lines. It locates the editor action table entry that corresponds to the record and then passes control to the specific output routine, which processes that record type, via an XCTL macro-instruction.

If the record to be processed is one which requires the use of an internal routine, this routine passes control to that internal routine via an XCTL macro-instruction.

The DUMP COMMENT Routine (IEGNM00A)

If the action router routine determines that the record to be processed is a DUMP COMMENT type, it passes control to the DUMP COMMENT routine (see Chart 59) via an XCTL macro-instruction. The DUMP COMMENT routine extracts the comment number from the prologue record of the DUMP COMMENT record-pair and uses it to find the comment in the editor action table. The comment is moved to the current print buffer and then written onto SYSPRINT by the line out and print routines, respectively. At the completion of this action, control is returned to the editor router routine.

The DUMP TABLE Routine (IEGNP00A)

When the action router routine determines that a DUMP TABLE record is being processed, it passes control to the DUMP TABLE routine (see Chart 59) via an XCTL macro-instruction. The DUMP TABLE routine examines the action table entry associated with this record to determine which table type is to be dumped and then applies available names to the respective logical sections of the block. Lengths and names of fields, and the sequence of fields in a table are contained within the routine. The symbolic location of the table, acquired from the address analyzer routine, is written onto SYSPRINT with the table, using the line out and print routines.

The TRACE STOP Routine (IEGPA00A)

If the record type being processed by the action router routine is a TRACE STOP record, the action router routine passes control to the TRACE STOP routine (see Chart 59) via an XCTL macro-instruction. The TRACE STOP routine examines the editor map (if it exists) to acquire the symbolic identification of the TRACE macro-instructions whose trace actions are halted by this TRACE STOP action. This information is moved to the current print buffer by the line out routine and written onto SYSTEST by the print routine. Control is returned to the editor router routine.

The TRACE Routine (IEGNV00A)

Except for TRACE STOP, all TRACE macro-instruction records encountered by the action router routine result in a control transfer to the trace routine. (See Chart 60.) This includes records generated by the TESTRAN interpreter's trace start routine. If the record is a trace flow or trace refer output, the problem program instruction that caused the record to be generated is indicated in the output record as it appeared in storage.

If the instruction was performed remotely by an execute instruction, it is indicated in its modified form along with the execute instruction. Registers involved and comments requested are also included in the output record generated and moved to the print buffer by the line out routine and written onto SYSPRINT by the print routine. The trace routine links to the address analyzer, attribute analyzer, dump data, and dump panel routines.

The TEST OPEN Routine (IEGPG00A)

The action router routine passes control to the test open routine (see Chart 61) when it encounters a test open type record as its input. The test open routine finds the action table entry that corresponds to the record and extracts the information about the TEST OPEN macro-instruction from that entry.

The line out routine moves the data to the current print buffer, the print routine writes the data onto SYSPRINT. When the test open routine is terminated, control returns to the editor router routine.

The DUMP MAP Routine (IEGNG00A)

If the action router routine detects a dump map type record, it passes control to the dump map routine (see Chart 59) via the XCTL macro-instruction. The routine takes the data in the record, which shows the storage configuration used by the program under test at its execution time, and writes it onto SYSPRINT.

The line out routine moves the data to the current print buffer, the print routine writes the data onto SYSPRINT. At completion, the DUMP MAP routine returns control to the editor router routine.

The TEST CLOSE Routine (IEGPH00A)

A test close type record causes the action router routine to pass control to the test close routine (see Chart 53) via the XCTL macro-instruction. This routine lists the TESTRAN interpreter TIA tables which the TEST CLOSE entry caused to be closed (deactivated). The routine uses the address analyzer routine to correlate the symbolic name of the tables with their storage addresses and uses the line out and print routines to move the data to the current print buffer and write it onto SYSPRINT, respectively.

The routine also adjusts the editor's internal tables to reflect the altered status of the testing environment of the problem program brought about by the test close operation. For each TIA table closed by the test close routine, its corresponding entry in the action table list is zeroed.

Exit from the test close routine is to the editor router routine via the RETURN macro-instruction.

The DUMP CHANGES Routine (IEGND00A)

The action router routine passes control to the dump changes routine (see Charts 62 and 63) via an XCTL macro-instruction when it encounters a dump changes type record. The routine uses the dump change list to determine if an earlier image of the storage area covered by this record exists. If such an image does exist, the dump change list points to its dump change table on secondary storage.

The dump change table entry indicates the address of the image, also on secondary storage. This image is compared byte-by-byte and field-by-field with the image contained within the input record. If no variations exist between the current (input) image and the one on secondary storage, "none" is written onto SYSPRINT. If, however, a difference does exist, the changed areas are written onto SYSPRINT. The new, or latest, image of the storage area becomes the new standard for comparison, and replaces the old image on secondary storage.

The dump changes routine passes control to other editor routines via the LINK macro-instruction. During its operation, this routine uses the symbol search routine, the attribute analyzer routine, the editor message routine, and the root module's line out and print routines.

Exit from the dump changes routine is to the editor router routine via a RETURN macro-instruction.

The DUMP DATA Routine (IEGNA00A)

The dump data routine (see Chart 61) is entered from the action router routine via an XCTL macro-instruction when the action router routine detects a dump data type

record or via a LINK macro-instruction from the trace routine for a trace refer type record. In either case, the routine produces output records that contain an image of a specified area of storage (at problem program execution time) with all possible labels and attributes indicated.

For a dump data record, this is the sole purpose, but for a trace refer record both a "before" and an "after" image must be produced. The routine scans the action table entry that corresponds to the interpreter TIA table entry whose interpretation caused the input record to be generated. It searches the entry for attribute overriders for the data. Any such overriders found are stored in the root module's communications area. The dump data routine links to the symbol search and attribute analyzer routines to locate and analyze symbol table attributes, respectively.

The data edit routines are called, as needed to process each field defined by the attributes.

Exit from this routine is via the RETURN macro-instruction. For the trace refer record, the return is to the trace routine; for the dump data record, it is to the editor router routine.

The DUMP PANEL Routine (IEGPP00A)

The dump panel routine (see Chart 61) receives control from the action router routine via an XCTL macro-instruction when the action router routine detects a dump panel type record. The record, as generated by the interpreter, contains images of all 16 general registers, the floating point registers (where applicable), and the program status word (PSW). The register selection mask in the action table is interrogated to determine which of the register images is to be printed.

The dump panel routine links to the data edit routines to convert the data in the registers to printable form and expands the PSW. The root module's line out and print routines are used to move the data to the current buffer and to write the data onto SYSPRINT, respectively.

Exit from the dump panel routine is to the editor router routine via the RETURN macro-instruction.

The Address Analyzer Routine (IEGSN00A)

The address analyzer routine (see Chart 64) is entered from the other output routines via a LINK macro-instruction. It identifies a given loaded address by its assembled address, its symbolic name, and its CSECT name. This routine obtains this information by searching the map, and supplies it to the print buffer via the line out routine, as requested by the calling routine. Following its execution, control passes to the calling routine via the RETURN macro-instruction.

The Symbol Search Routine (IEGSQ00A)

The symbol search routine (see Chart 64) is linked to by the other editor routines. It finds the attributes of a data item for which the calling routine can supply the loaded address. The routine searches the editor map and the corresponding symbol table for these attributes and sets a pointer to them.

If the loaded address supplied for a data item is not at the beginning of that item, but at some offset from the beginning, the symbol table attributes (length, duplication factor, etc.) are altered to reflect this. For duplicated items, the duplication factor is altered to properly reflect the portion of the item from the offset to the end of the item. For multiple DSECTs, an offset from the beginning of the first DSECT is converted to an actual offset into the correct copy.

The attributes for the data, as determined by this routine, are made available to the calling routine.

Exit from the symbol search routine is via a RETURN macro-instruction to the calling routine.

The Attribute Analyzer Routine (IEGSR00A)

The attribute analyzer routine (see Chart 65) is linked to by other output routines (dump data, dump changes, and trace.) This routine correlates the attributes of a data item given to it at assembly time or overriders provided by the user with the data item. It maintains the relationship, between data and its attributes, that was established by the symbol search routine. It does this by continually updating pointers so that the proper attributes continue to be applied to data

and by causing the data to be converted to hexadecimal format when input data comes from the subsequent CSECTs. The routine also initiates the writing of a print buffer when a line is filled and identifies each print line with its assembled and loaded addresses. Attribute pointers are maintained in the root module's communication area. This routine links to the appropriate data edit routine through its edit linkage subroutine (IEGEDIT).

Note: The TESTRAN editor uses a system of three print line buffers. The format of the printed output for a specific type of test output data can be found in the publication IBM System/360 Operating System: Control Program Services.

The Data Edit Routines

The data edit routines consist of eight routines that convert data from one format to another and prepare it for printing. These routines are linked to either by the attribute analyzer routine or by the DUMP PANEL routine. The calling routine remains resident in main storage during the operation of a data edit routine.

If a record being processed has a continuation record, control passes from the data edit routine to the editor router. The editor router gets the continuation record and passes control back to the data edit routine. The amount of data converted by an edit routine is dependent on the item's length attribute as stored in IEGCMARA within the communications area of the root module. All results from data edit routines are in the EBCDIC print character form. The actual moving of edited data from the work area to the print buffers is accomplished by the attribute analyzer routine in all edits except zoned decimal, alphameric and hexadecimal (over 4 bytes).

Hexadecimal Edit Routine (IEGSU01Z): The hexadecimal edit routine (see Chart 66) converts 4-bit hexadecimal values to the corresponding EBCDIC character. The routine moves the substituted EBCDIC characters to the current print buffer and returns control to the calling routine. The processing is done one field at a time, with control returning to the calling routine after each field is processed.

Instruction Edit Routine (IEGSU06Z): The instruction edit routine (see Chart 66) replaces the operation code portion of an instruction image (in its internal machine coded form) with the assembler language mnemonics for that instruction. The operand portion of the instruction image is converted to the EBCDIC characters for the hexadecimal values they represent. If a TESTRAN SVC instruction, that was inserted by the interpreter into the user's program, is encountered, its image is placed into the print buffer so that it appears directly below the instruction bytes it displaces when the buffers are printed.

The correlation between SVC and displaced instruction bytes is made through the use of the editor's reference table. If no reference table exists, only the SVC instruction will be printed. If the routine encounters an invalid operation code byte, it is placed into the print buffers in its 2-hexadecimal-character form with the two characters placed side by side and flanked with asterisks in the printed output. The routine will retain control for the area defined by one entry in the symbol table, or for any area with an instruction overrider.

Alphameric Edit Routine (IEGSU40Z): The alphameric edit routine (see Chart 66) moves a specified number of EBCDIC coded alphameric characters into the current print buffer and checks to determine whether or not each character is a valid print character. If a character is not a valid print character, this routine converts it to its 2-hexadecimal-character equivalent and places the two hexadecimal characters into the print buffers so that they will be printed one above the other.

The routine is linked to either by the dump panel routine or by the attribute analyzer routine. At the conclusion of its operation, control returns to the calling routine.

Binary Edit Routine (IEGSU50Z): The binary edit routine (see Chart 67) converts each bit within each specified byte into its equivalent decimal value, and moves the decimal character (or characters) into the current print buffer. The routine leaves the result of its conversion of the last specified byte in the work area. The calling routine moves the last byte's converted value into the print buffer.

Zoned Decimal Edit Routine (IEGSU60Z): The zoned decimal edit routine (see Chart 66) checks the validity of the sign of the zoned decimal number to be processed. If it is a valid sign, this routine places that sign ahead of the number in the print buffer. The digits are moved to the print buffer and any found to be invalid are converted into their 2-hexadecimal-character equivalent and placed into the print buffer so that they are printed one above the other.

If the length of the input data field is greater than 16 bytes, control passes to the alphameric edit routine via an XCTL macro-instruction. Otherwise, control returns to the calling routine.

Packed Decimal Edit Routine (IEGSU70Z): The packed decimal edit routine (see Chart 66) converts or "unpacks" data in packed decimal form into a zoned decimal equivalent. This routine does not perform a validity check of the data. If the length of the input field is greater than 16 bytes, control passes to the hexadecimal edit routine via an XCTL macro-instruction.

Fixed-Point Edit Routine (IEGSU80Z): The fixed-point edit routine (see Chart 51) converts binary numbers to decimal numbers. This routine operates on binary numbers with a scale that is equal to, greater than, or less than zero. Processing of the number by the routine depends on the scale, with a different procedure for the 3 variations in scale. The converted output (in the form of decimal numbers) is placed into the print buffer. Control then returns to the calling routine.

Floating-Point Edit Routine (IEGSU90Z): The floating-point edit routine (see Chart 65) converts a floating-point number in its internal hexadecimal coded form into a decimal number and places the decimal number generated into the current print buffer. The actual conversion routine follows several basic steps, some of which depend on whether the floating-point number is short or long. The relative scale of the number also affects the processing flow. More processing is necessary for larger or smaller numbers than for mid-range numbers.

It is possible that the separate job step of editing TESTRAN test output data may also be done on a separate computing system. It is possible that a computing system without the floating point feature might be called upon to edit floating point data. For this reason, only the standard instruction set is used in the floating point edit routine.

TESTRAN EDITOR CONTROL FLOW FOR SAMPLE EDIT

Figure 5 depicts the control flow through the TESTRAN editor for a sample-case editor run. The sample is a relatively simple one in which the problem program was tested with TEST OPEN, TEST AT, and TEST CLOSE test control macro-instructions and a DUMP DATA test action macro-instruction.

As Figure 5 indicates, initial entry to the TESTRAN editor is made into the start routine. After it initializes the program, the start routine transfers control to the editor router routine. This routine reads the input records from SYSTEST and passes control to the action router unless the record is a continuation record. For a continuation record, the editor router passes control directly back to the routine which had been processing the previous portion of the record.

When the action router gets control it examines the record type to determine which routine is to be given control. The records to be processed in this sample edit are indicated within the action router block, sequentially from top to bottom, in the succession in which they are read.

Opposite each record-type name are blocks representing the routines used to process that record type. In some cases, a routine in the longer sequences may transfer control back to a previous routine which is still resident in main storage. In other instances a routine that was used, early in the processing sequence, has to be loaded into main storage again.

Figure 5 shows that the records processed are:

Symbol Table
CESD
Relocation Table
TIA Table
Reference Table
Test Open
Dump Data
Test Close
End of Data

The end-of-data processing is initiated by an end-of-data-set condition rather than a record.

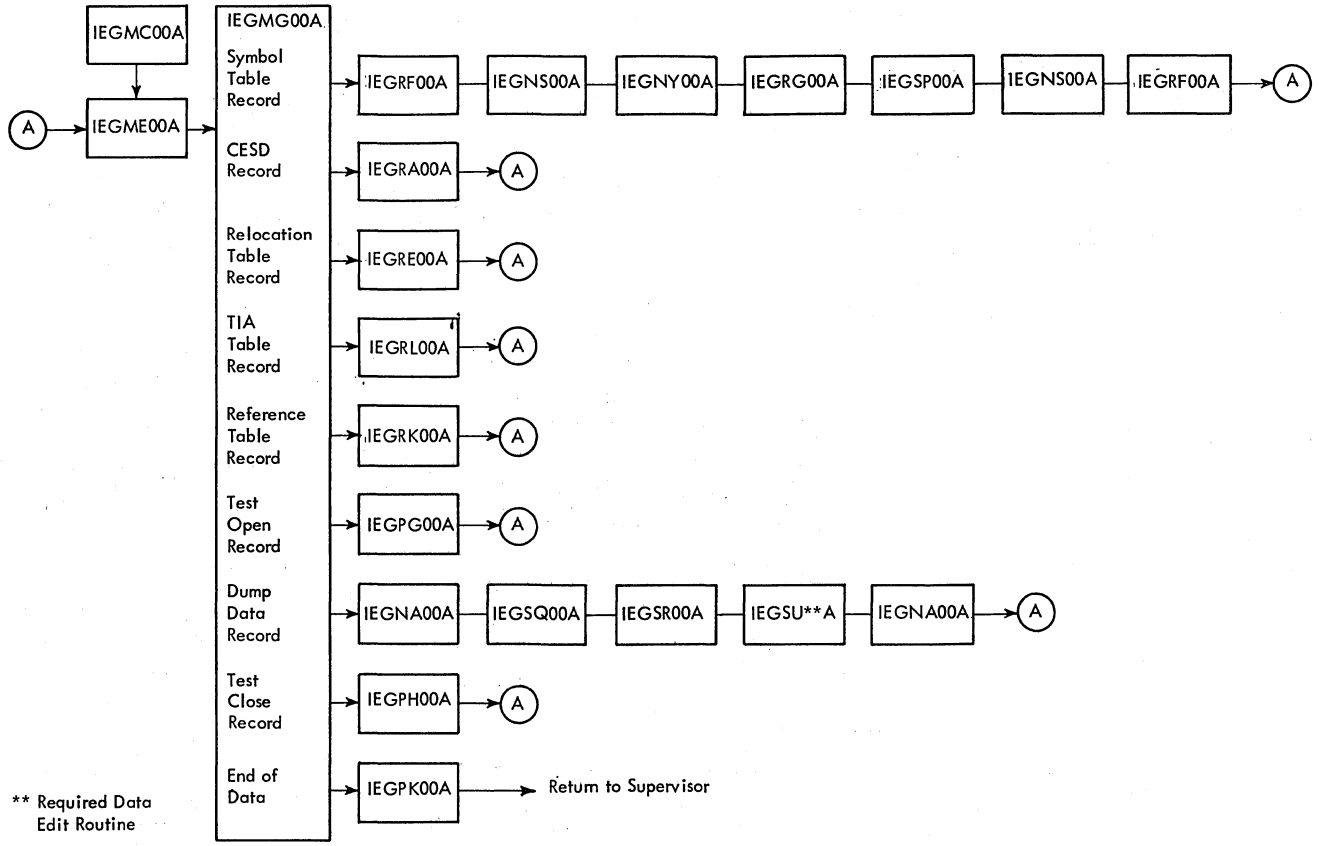


Figure 5. TESTRAN Editor Control Flow for Sample Edit

Flowchart representations of the routines described in the text are printed on the following pages. The numbering system used to identify the charts is as follows:

10 through 15 - TESTRAN macro-definition processes.

30 through 43 - TESTRAN interpreter routines.

50 through 67 - TESTRAN editor routines.

Chart 10. TESTRAN Macro-Definition - TEST

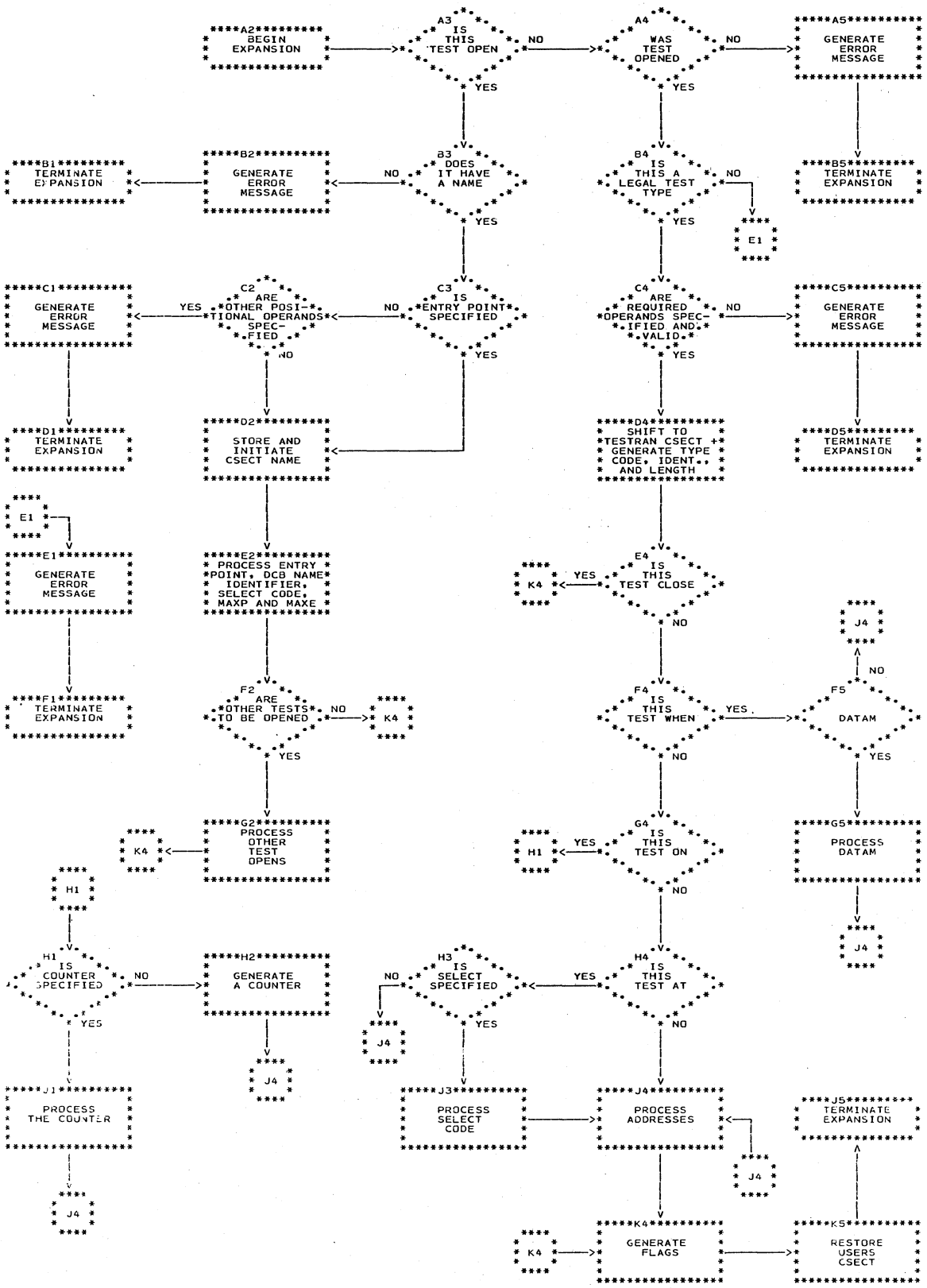


Chart 11. TESTRAN Macro-Definition - SET

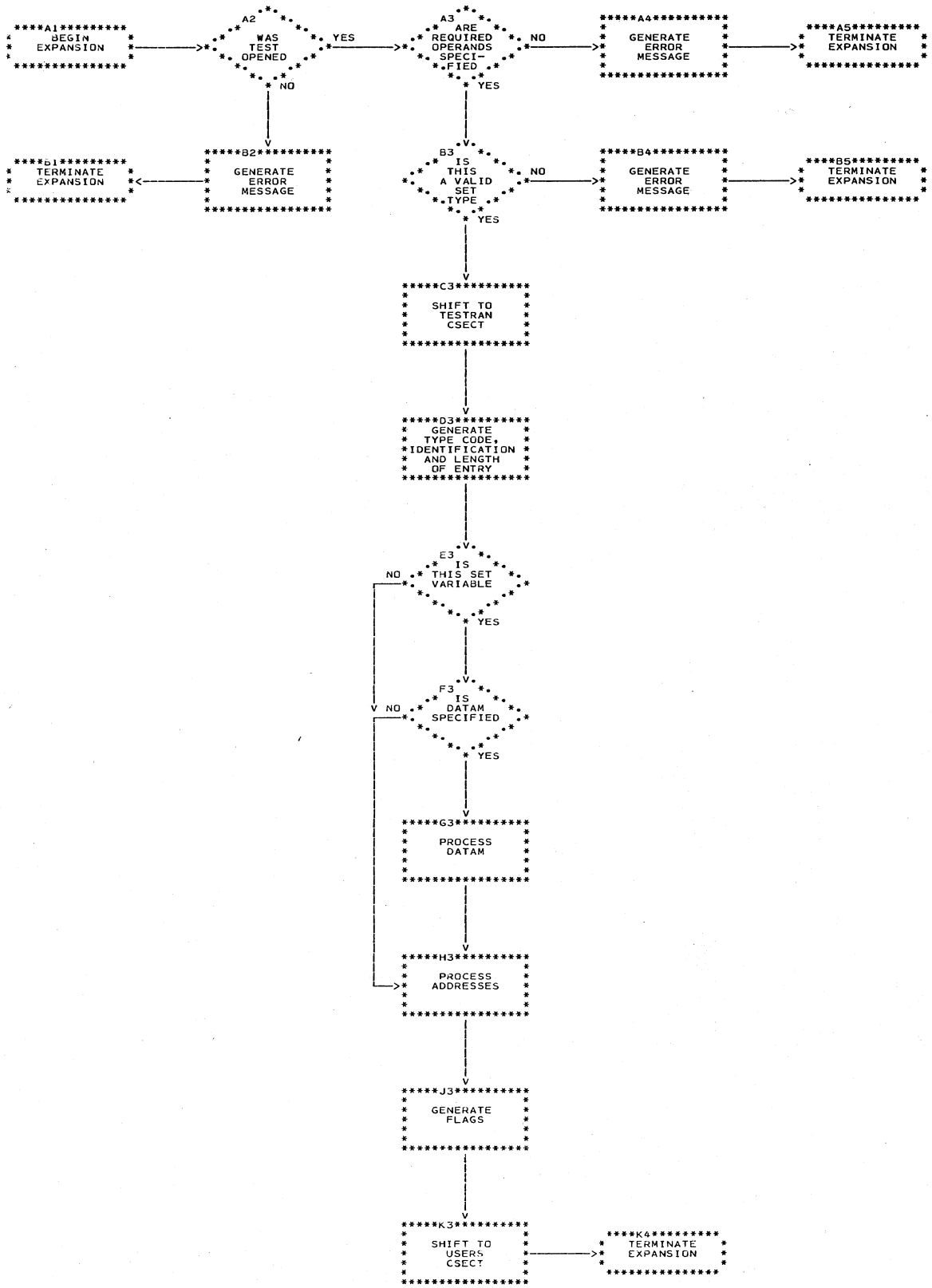


Chart 12. TESTRAN Macro-Definition - GO

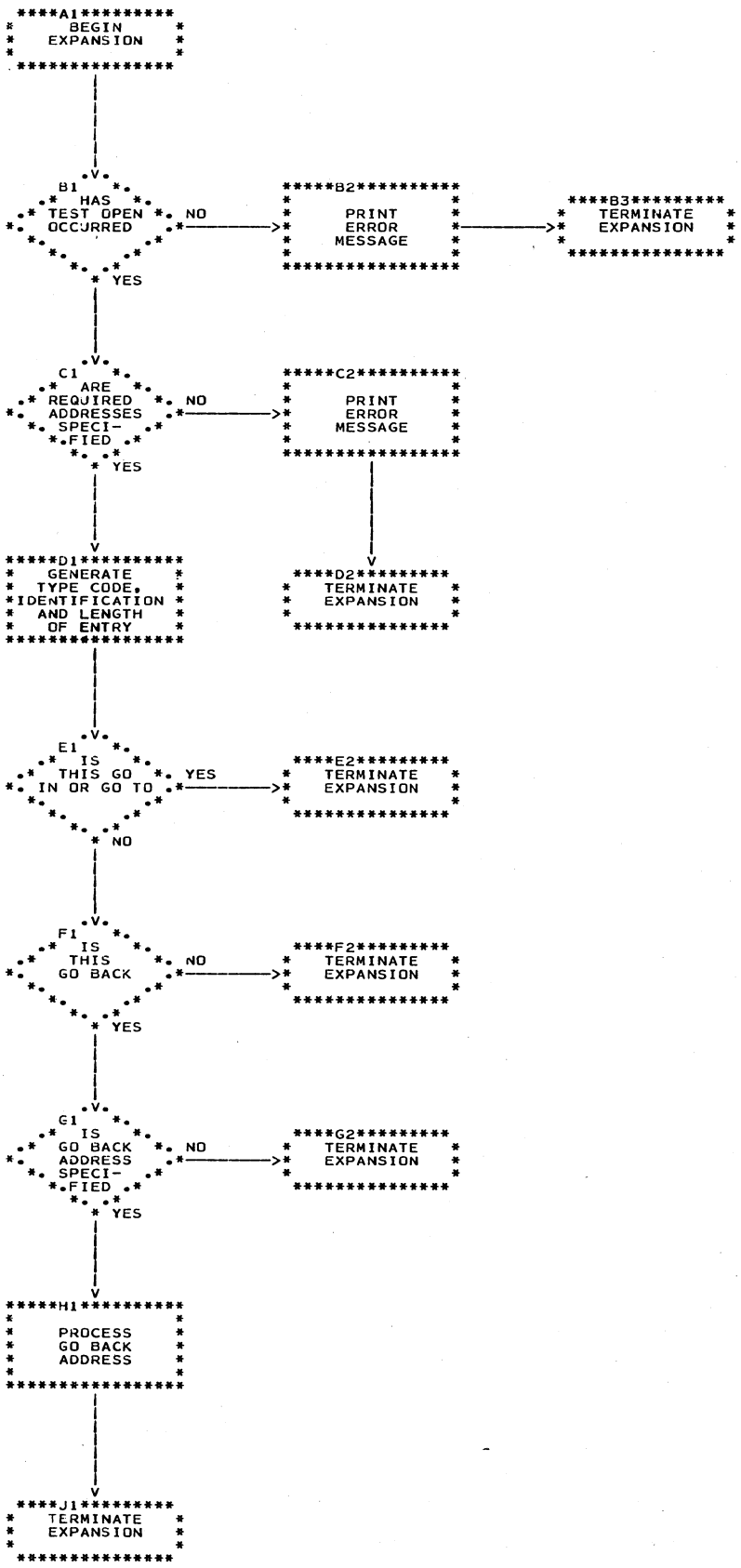


Chart 13. TESTRAN Macro-Definition - DUMP

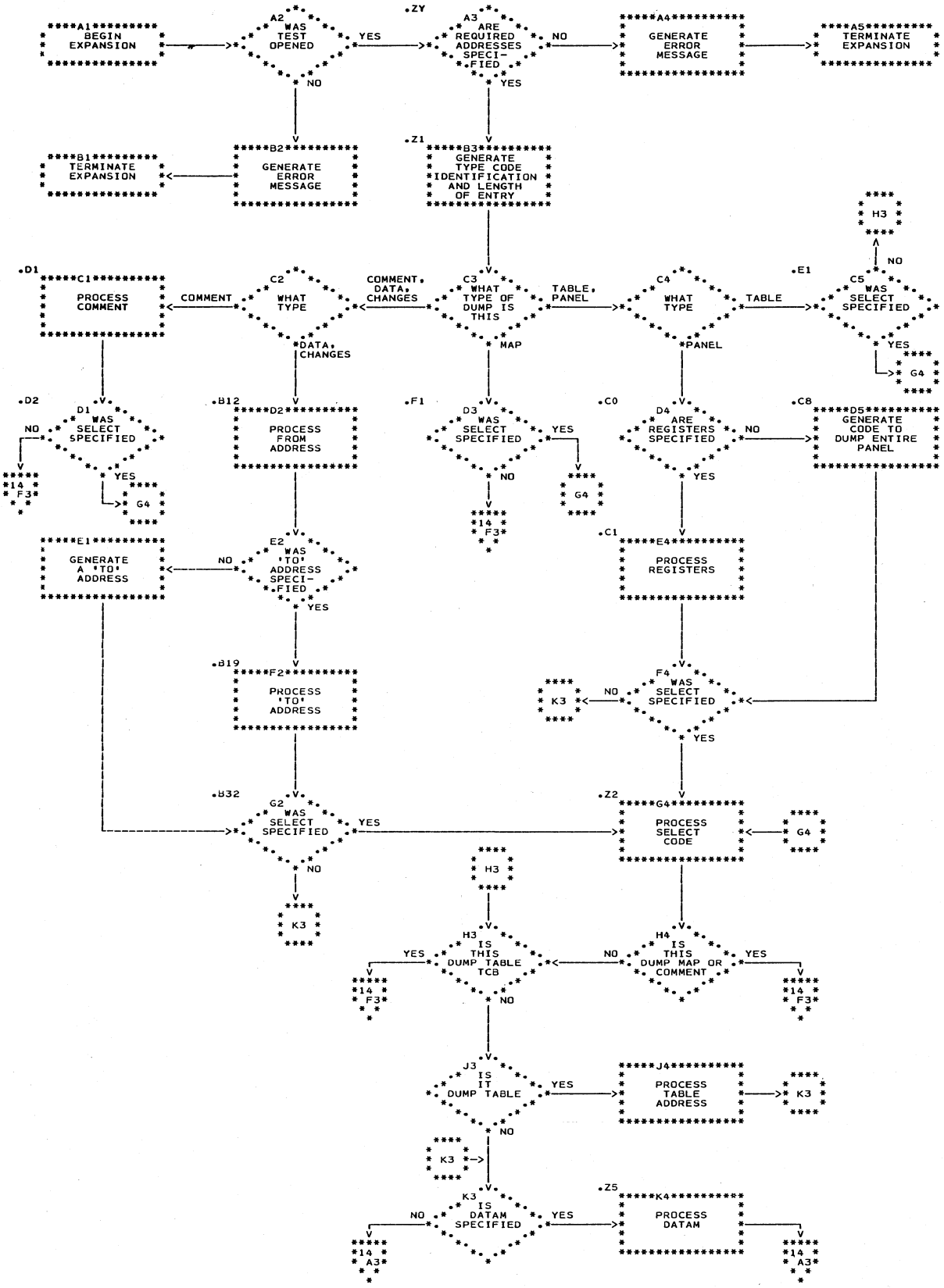


Chart 14. TESTRAN Macro-Definition - DUMP

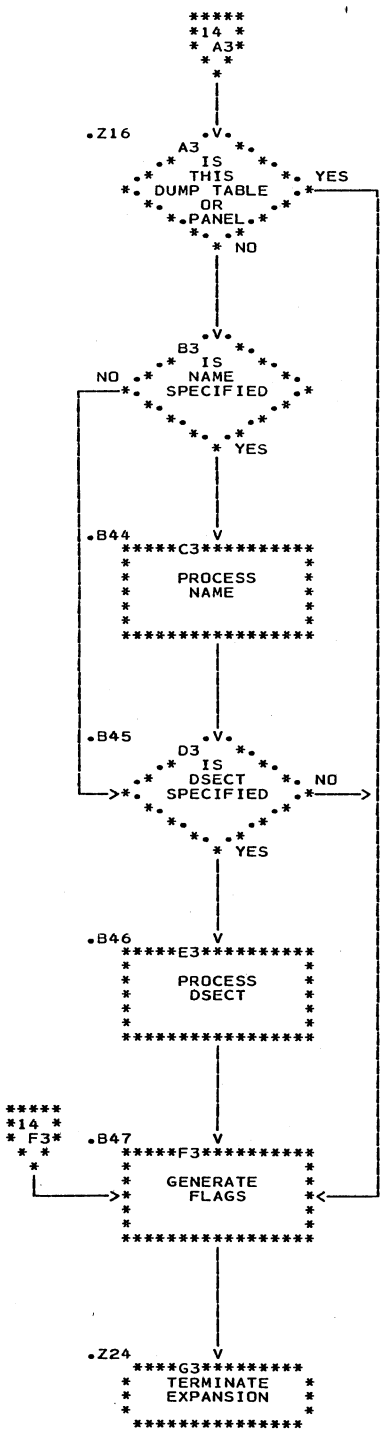


Chart 15. TESTRAN Macro-Definition - TRACE

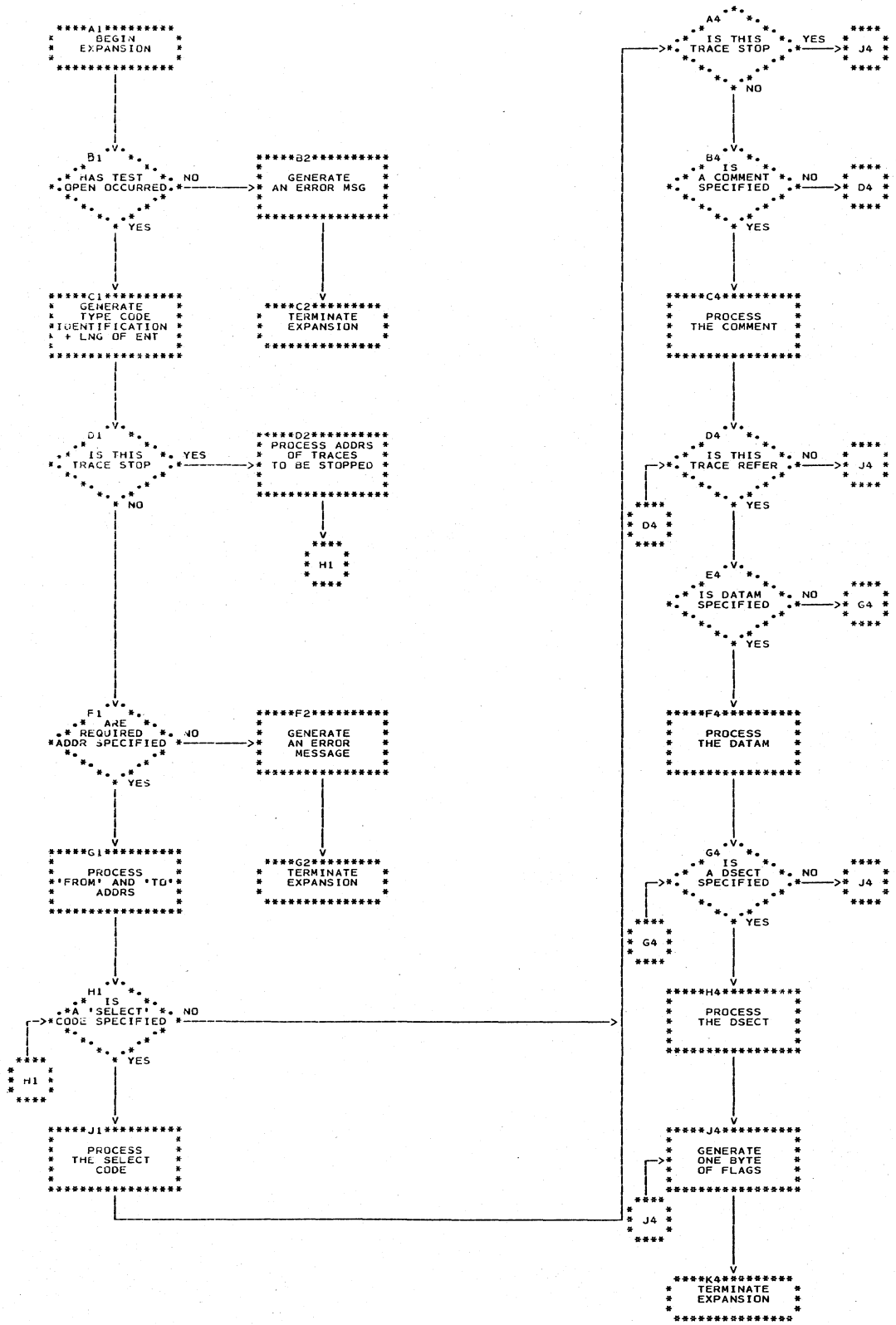


Chart 30. TESTRAN Interpreter

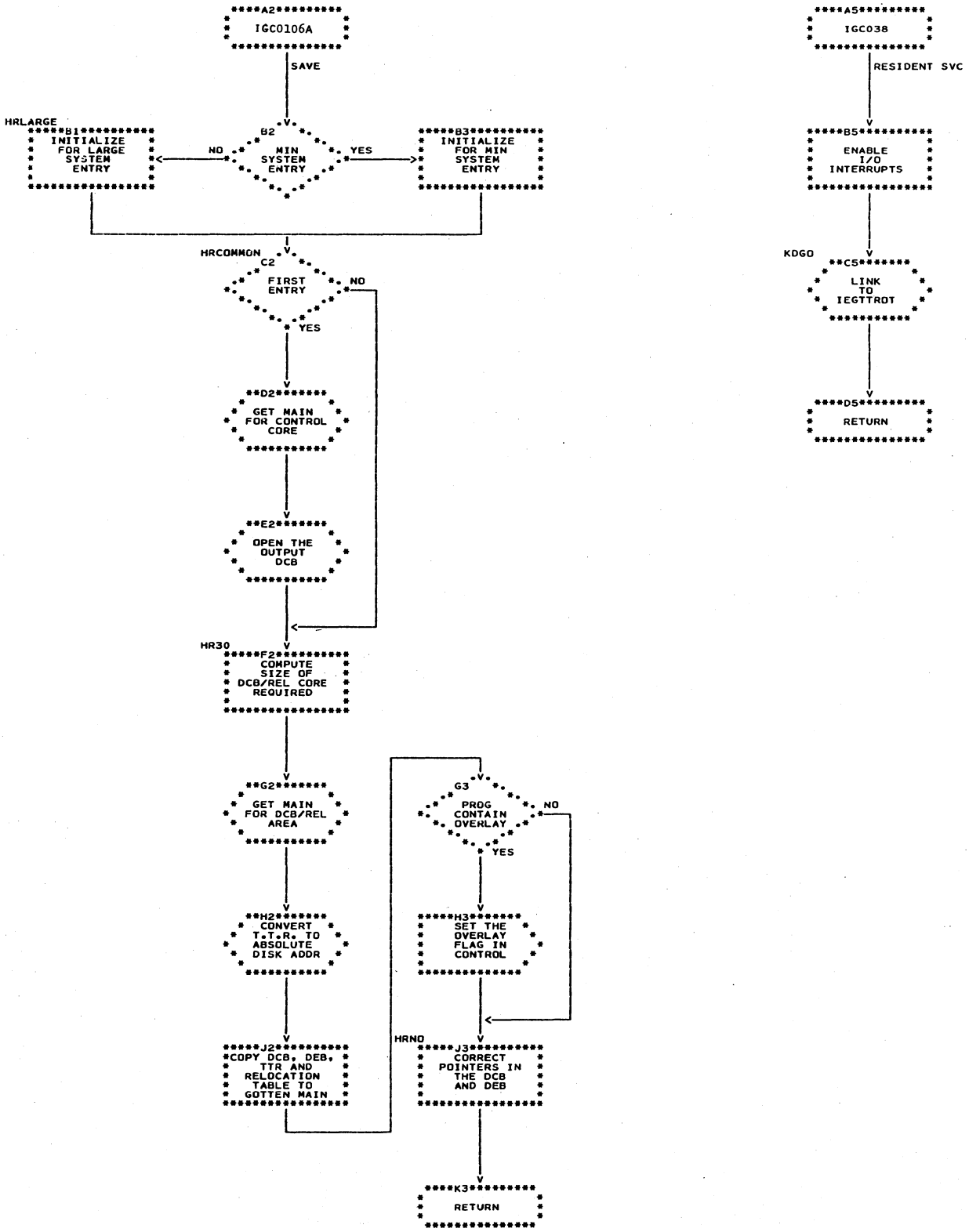


Chart 31. TESTRAN Interpreter

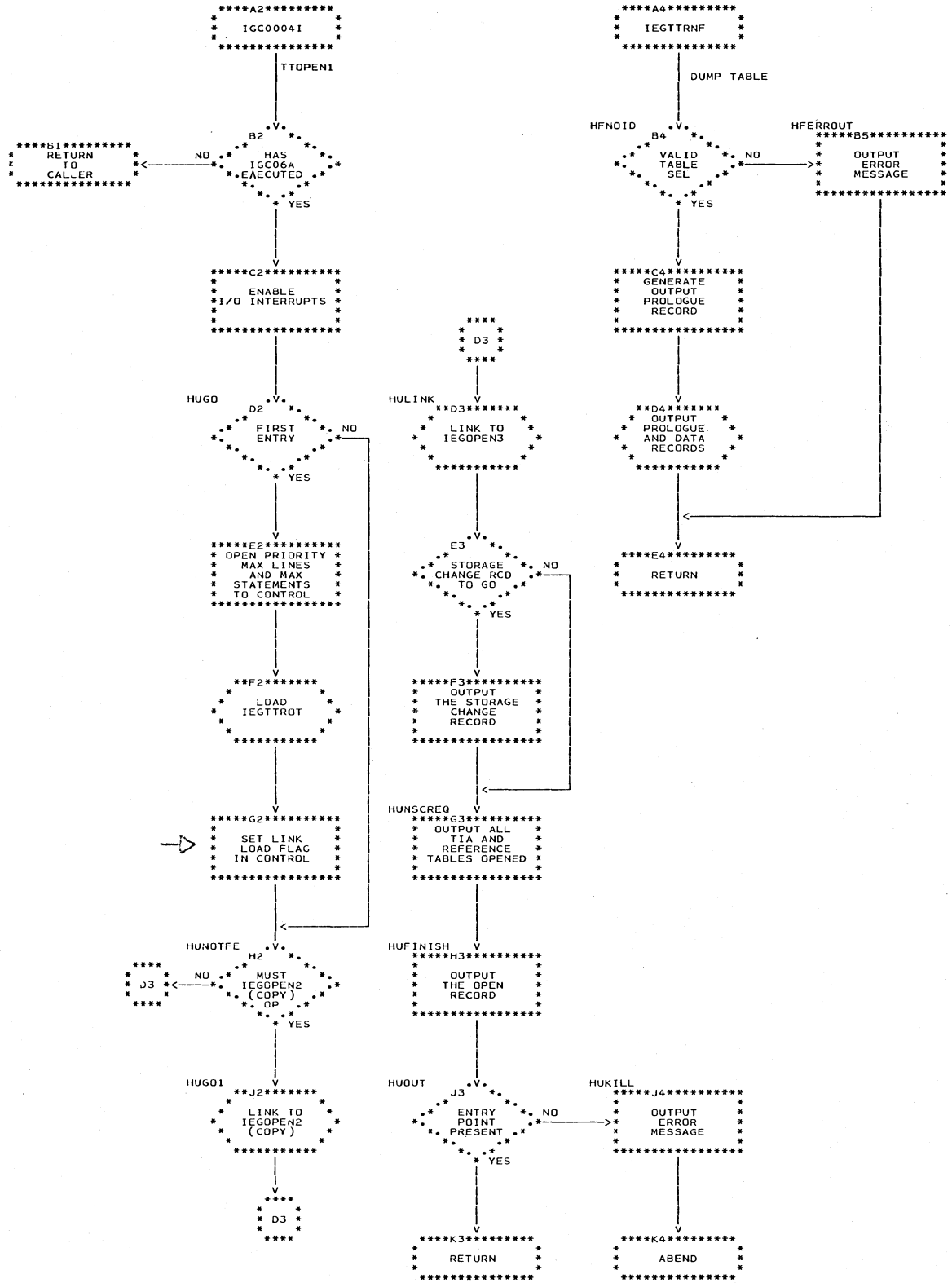


Chart 32. TESTRAN Interpreter

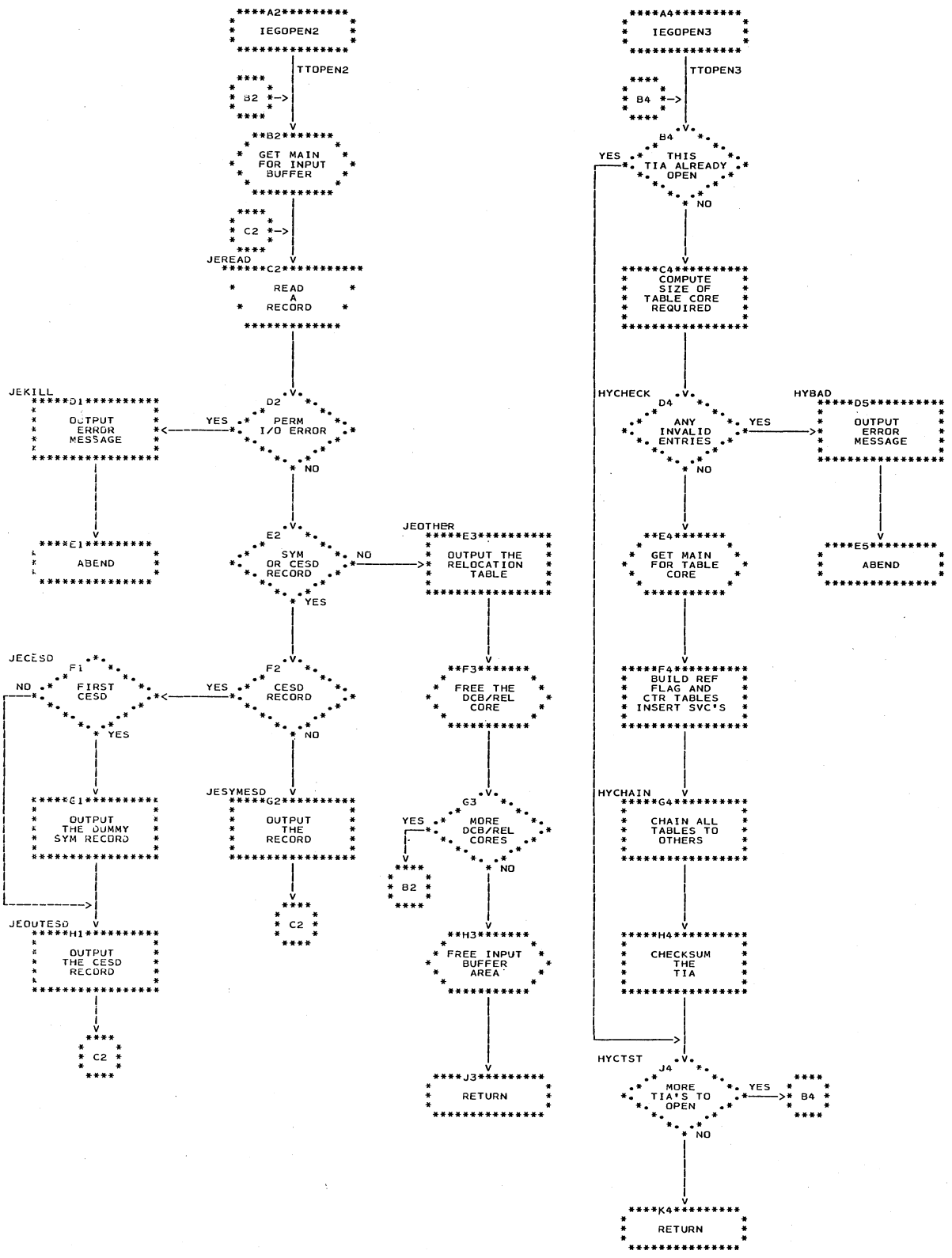


Chart 33. TESTRAN Interpreter

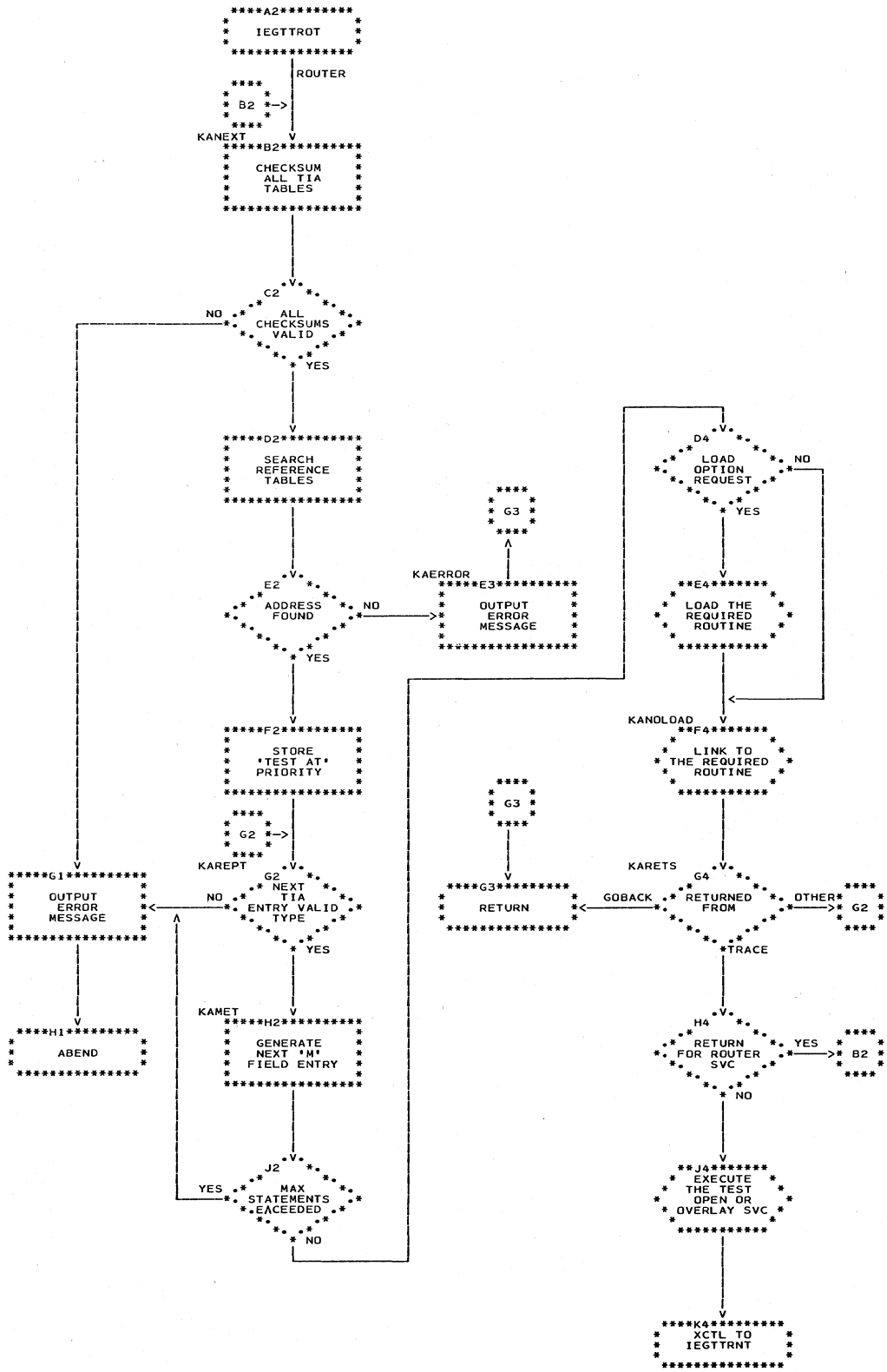


Chart 34. TESTRAN Interpreter

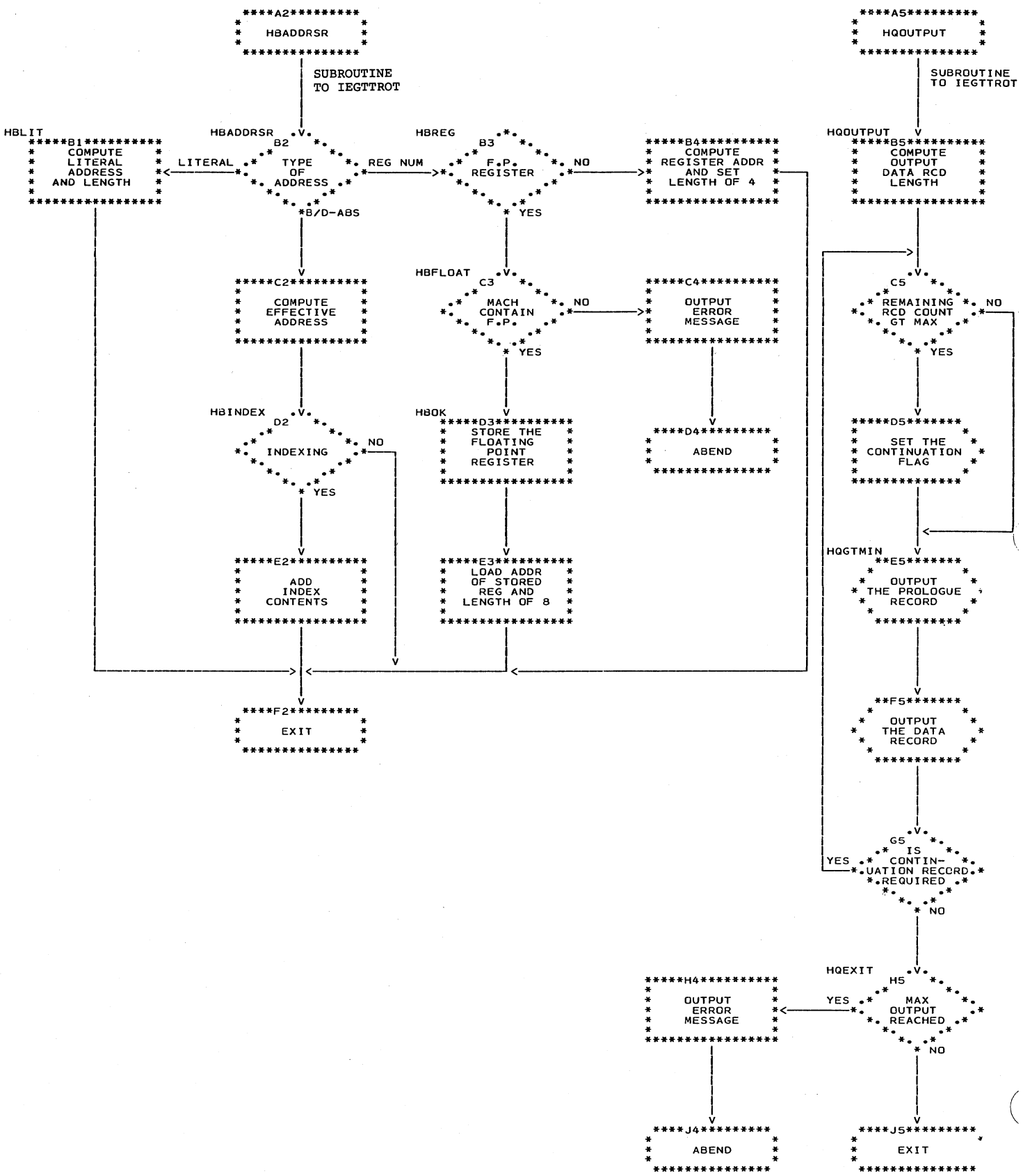


Chart 35. TESTRAN Interpreter

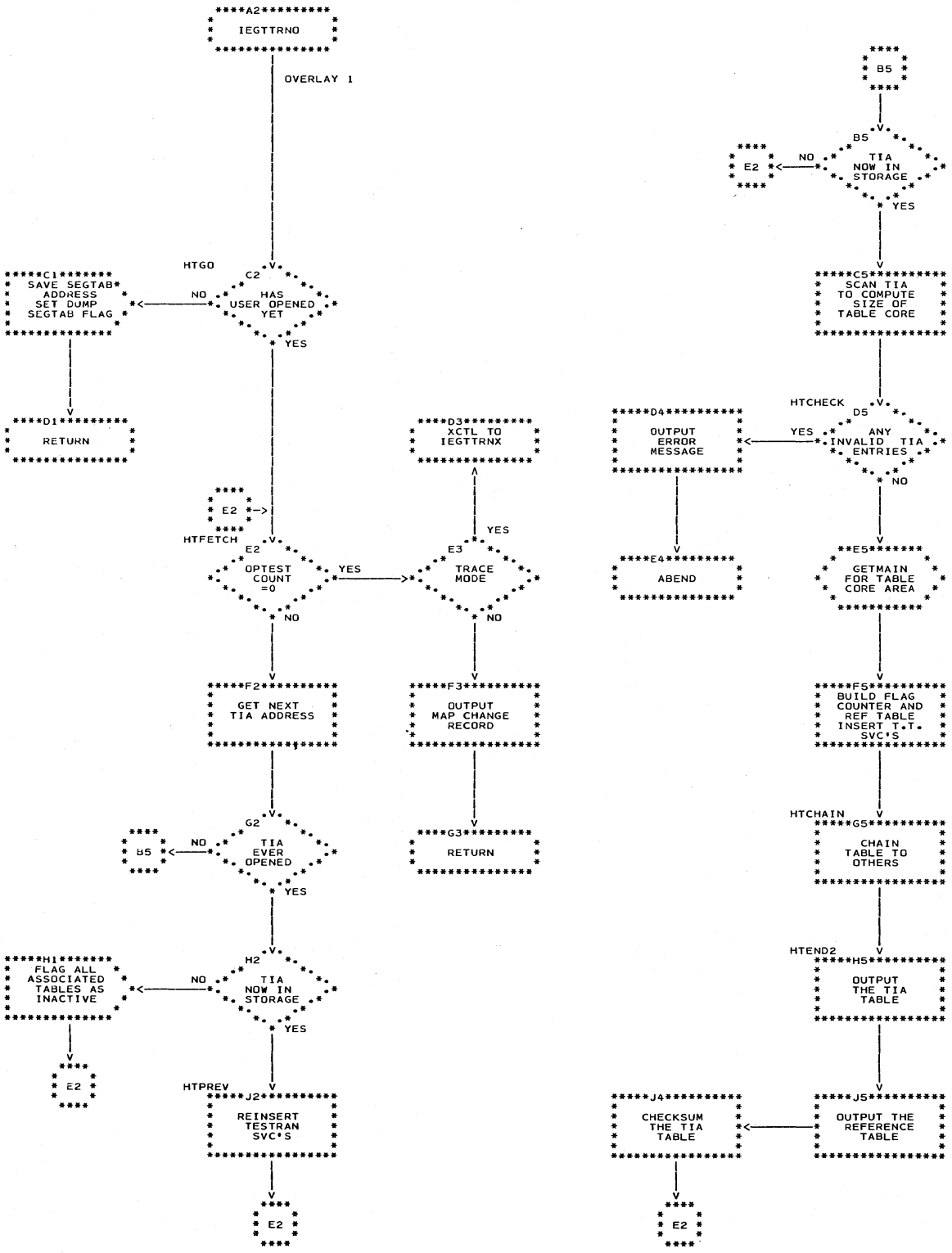


Chart 36. TESTRAN Interpreter

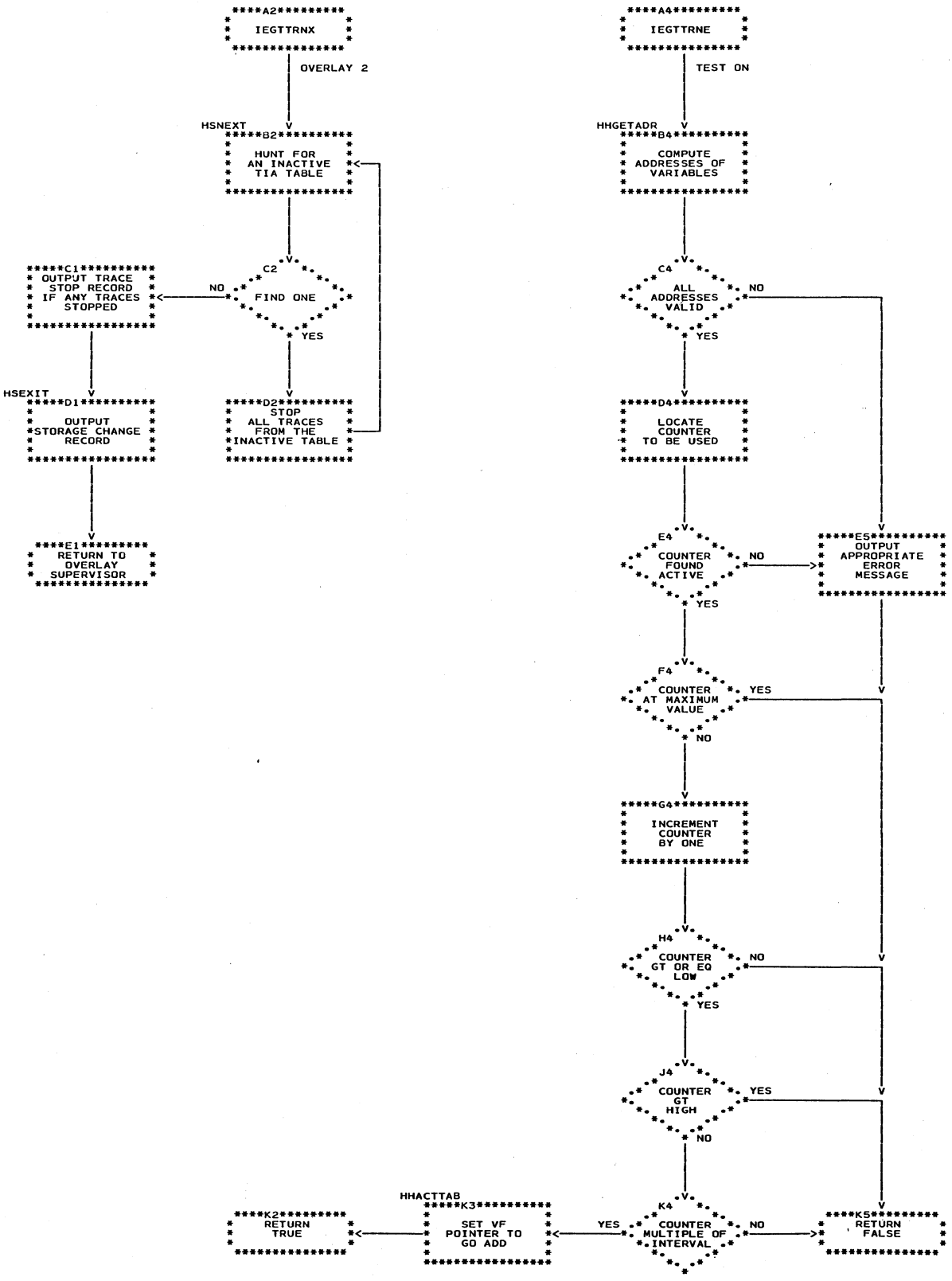


Chart 37. TESTRAN Interpreter

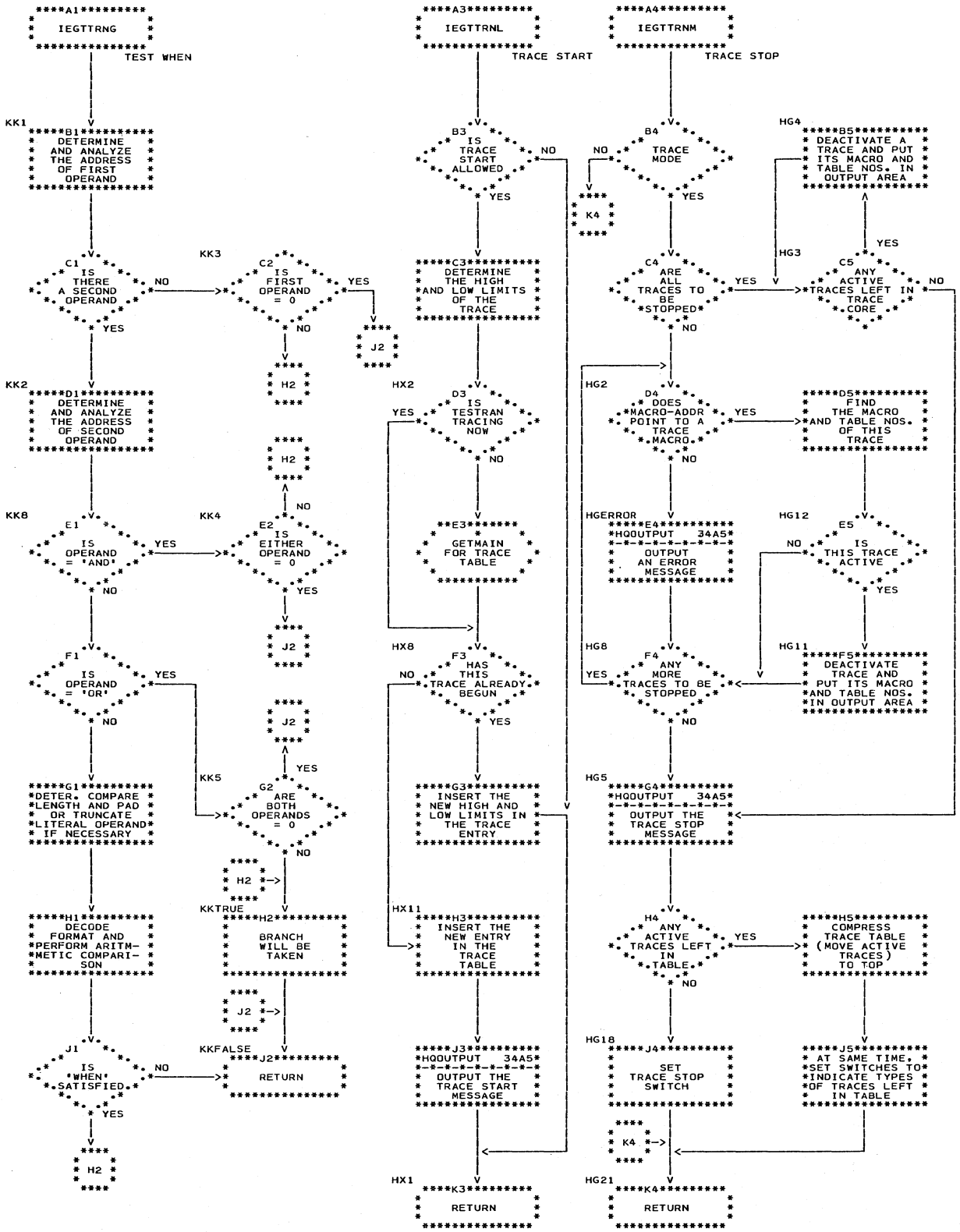


Chart 38. TESTRAN Interpreter

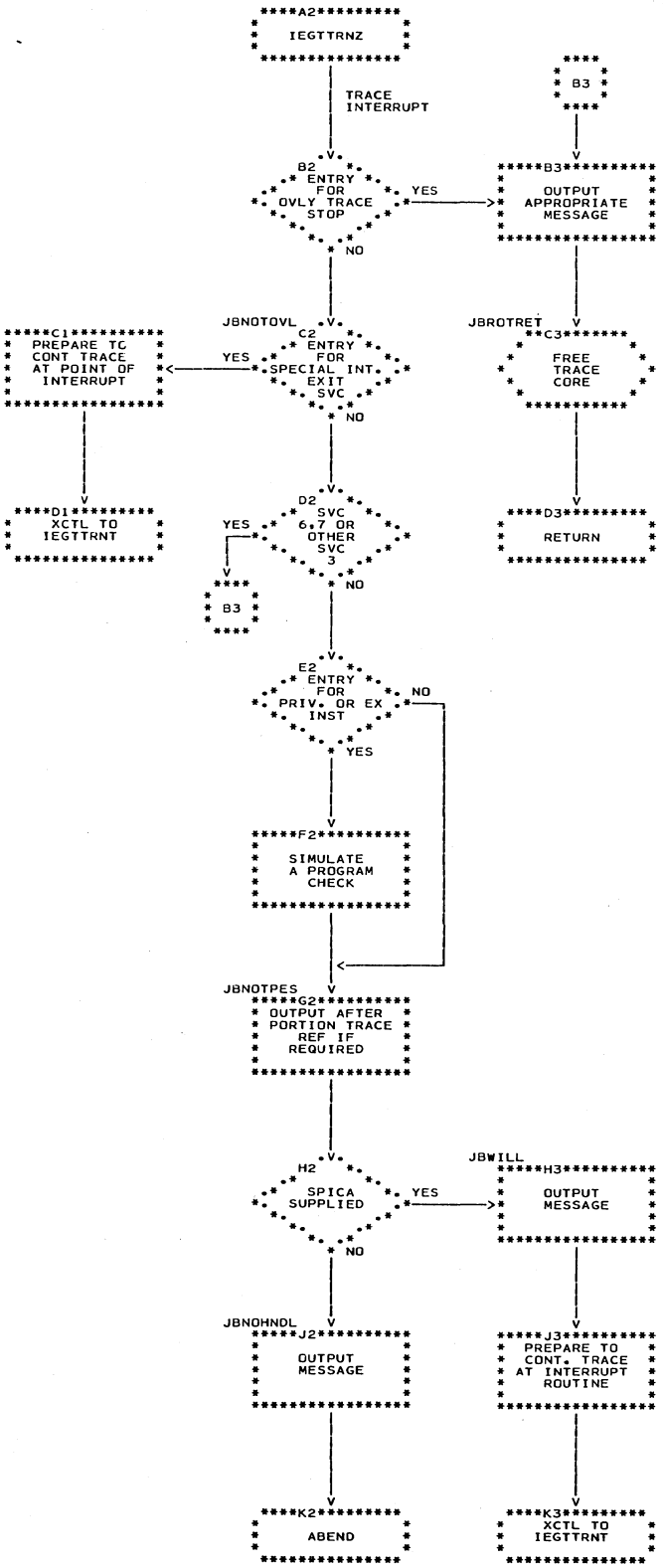


Chart 39. TESTRAN Interpreter

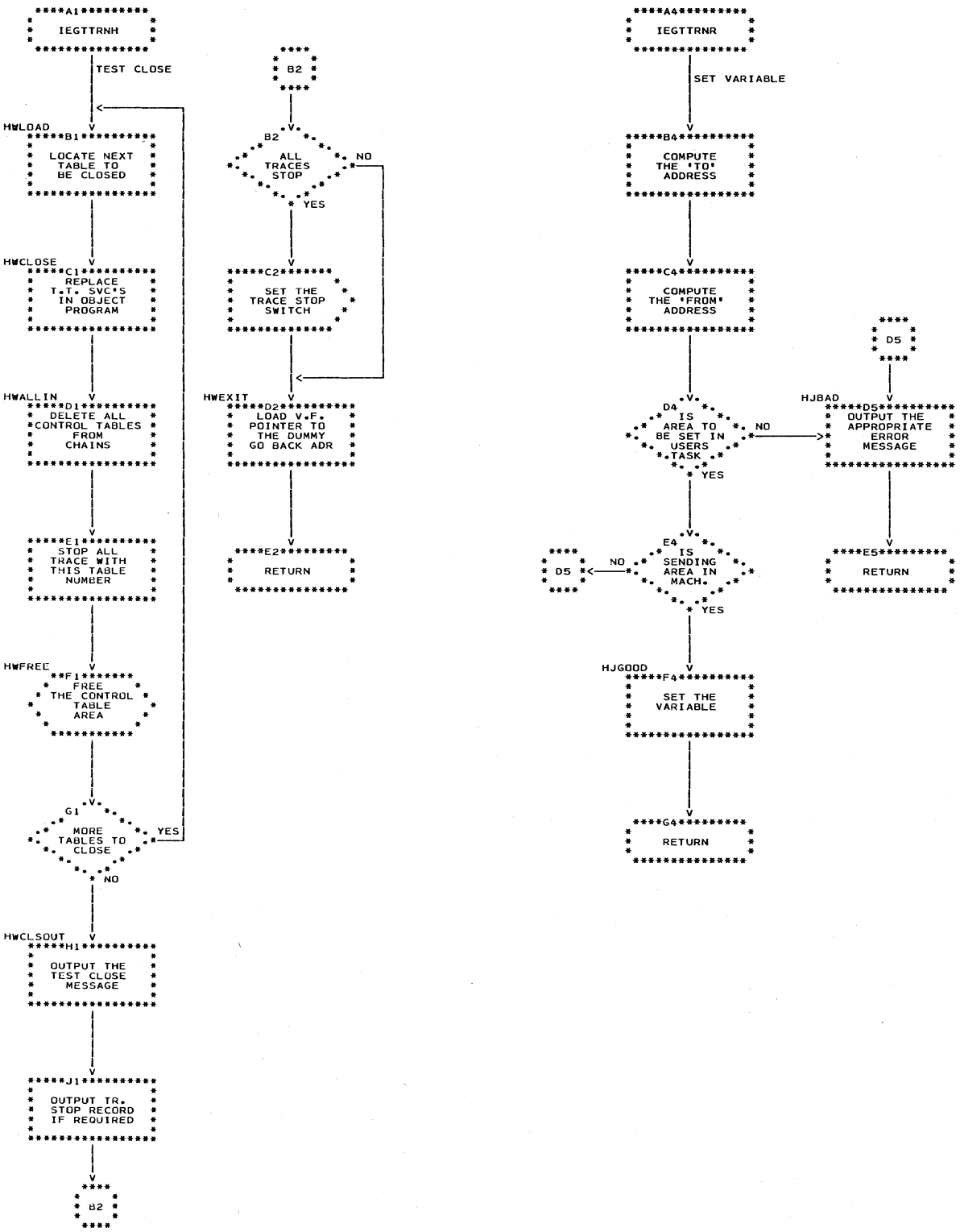


Chart 40. TESTRAN Interpreter

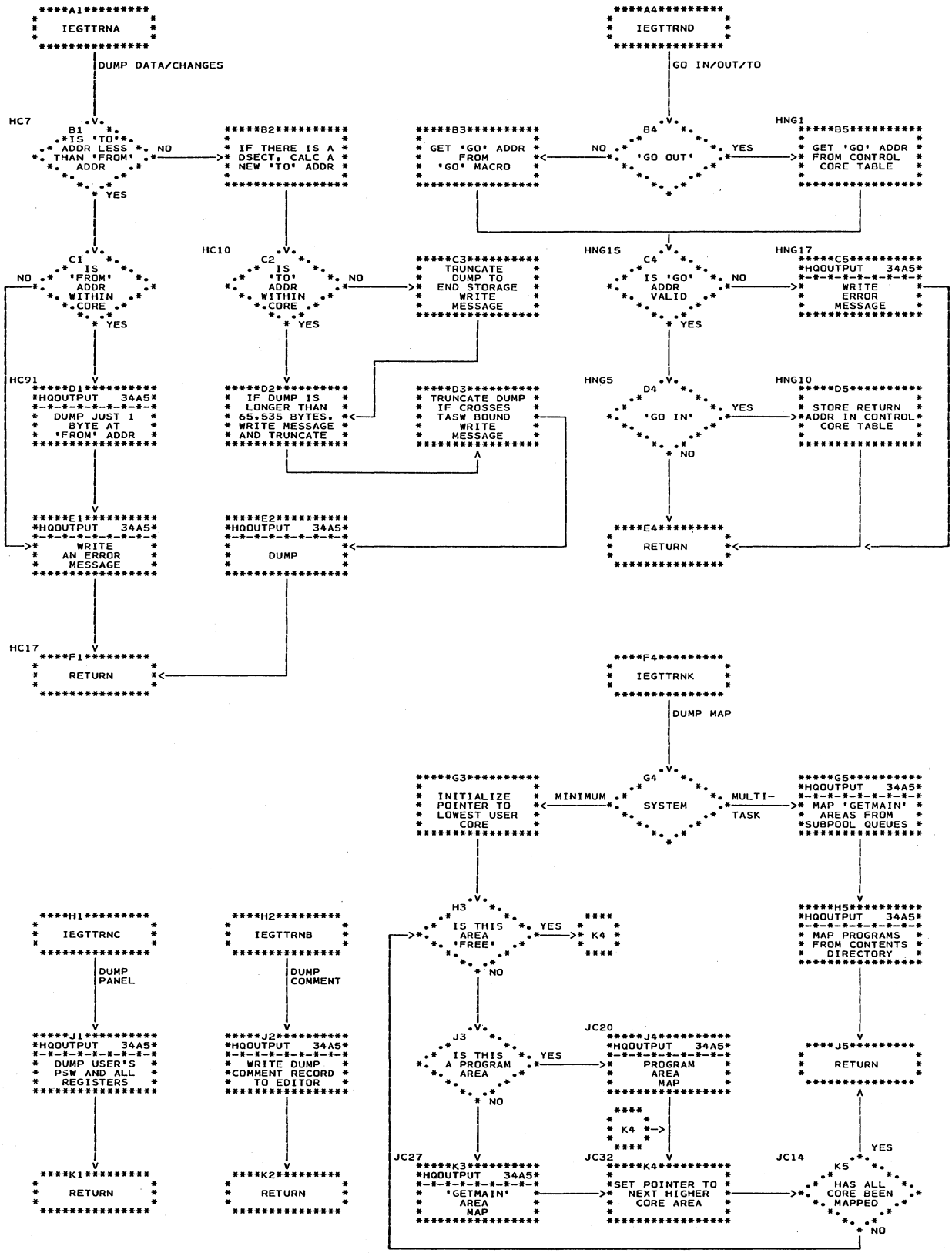


Chart 41. TESTRAN Interpreter

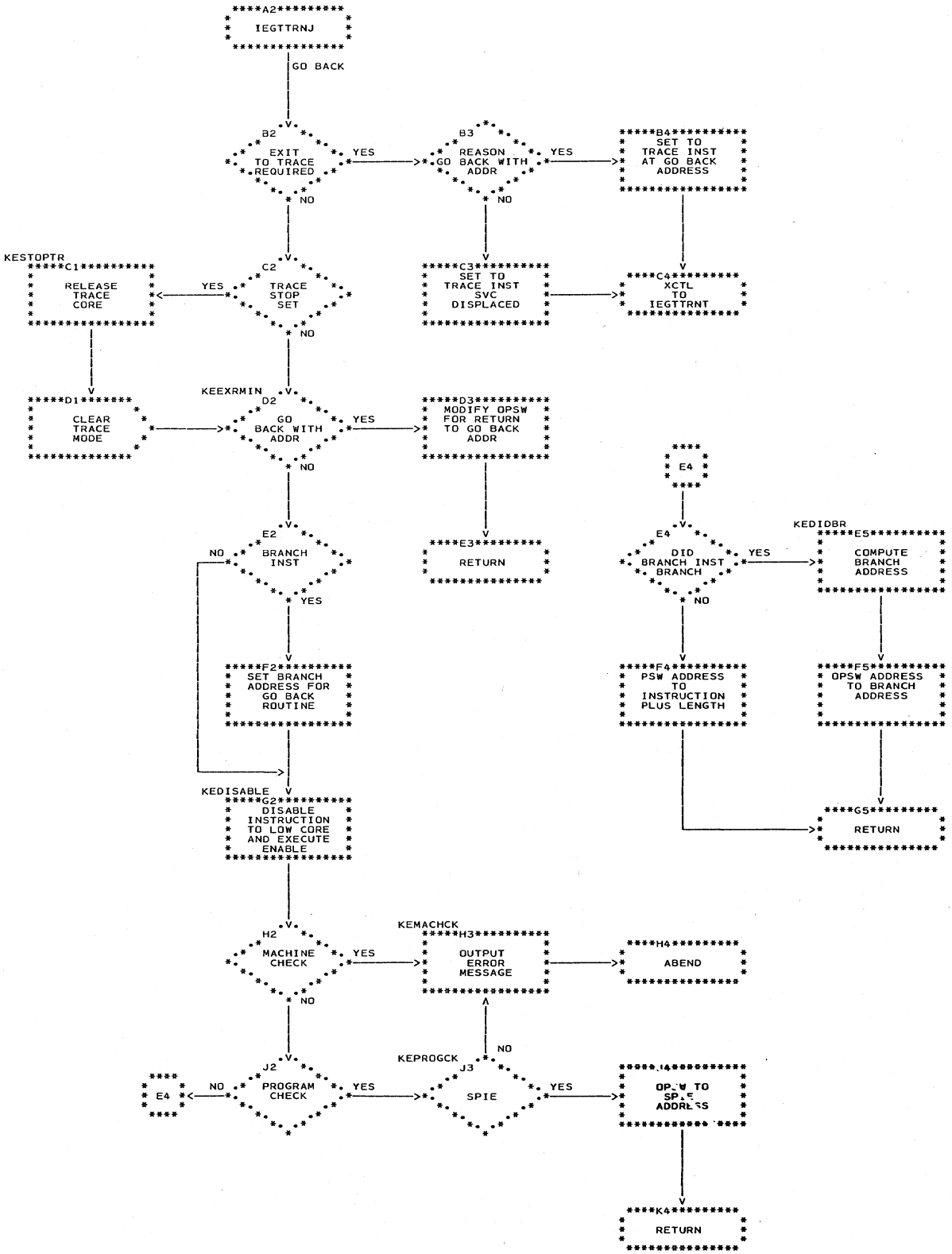


Chart 42. TESTRAN Interpreter

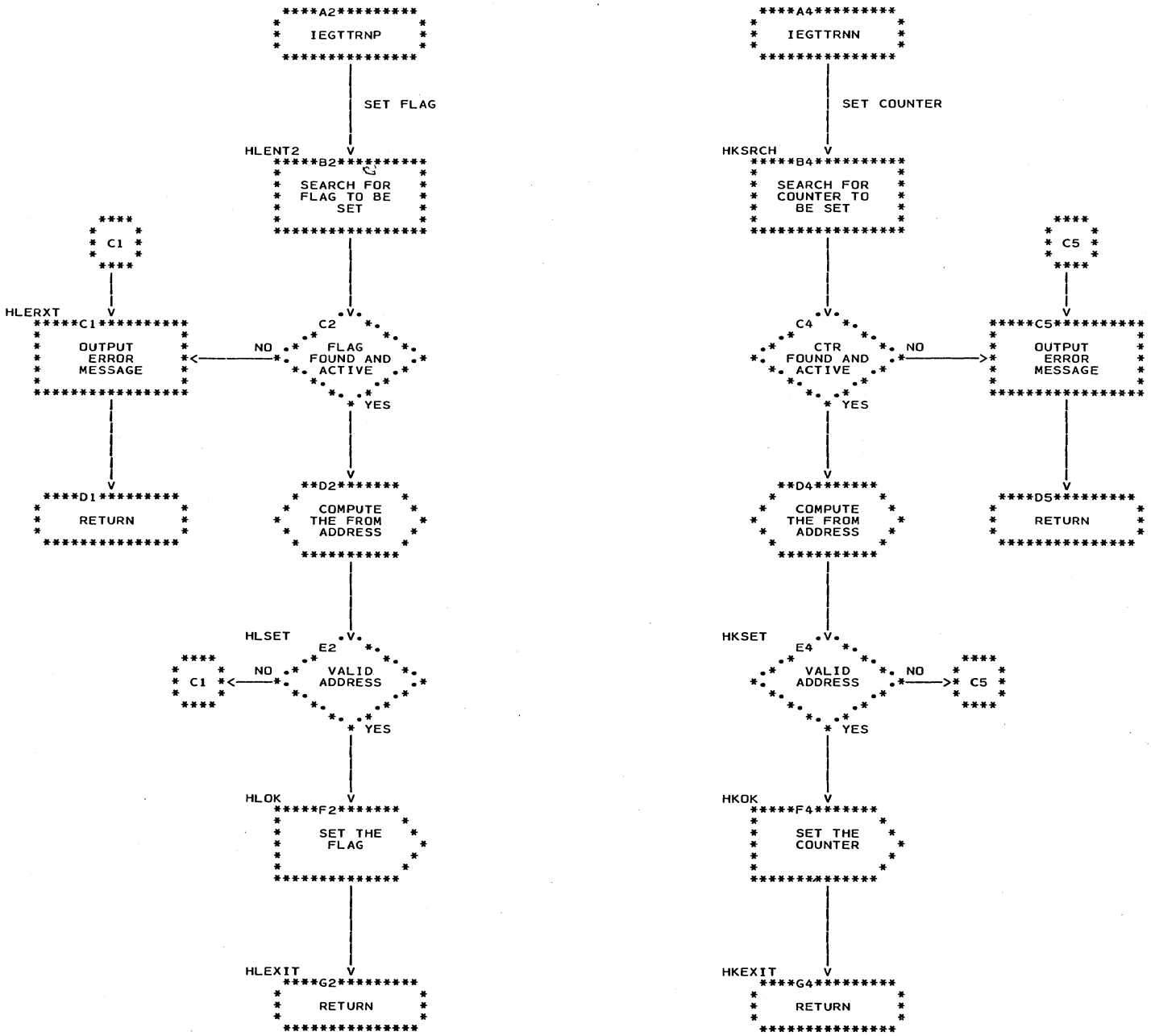


Chart 43. TESTRAN Interpreter

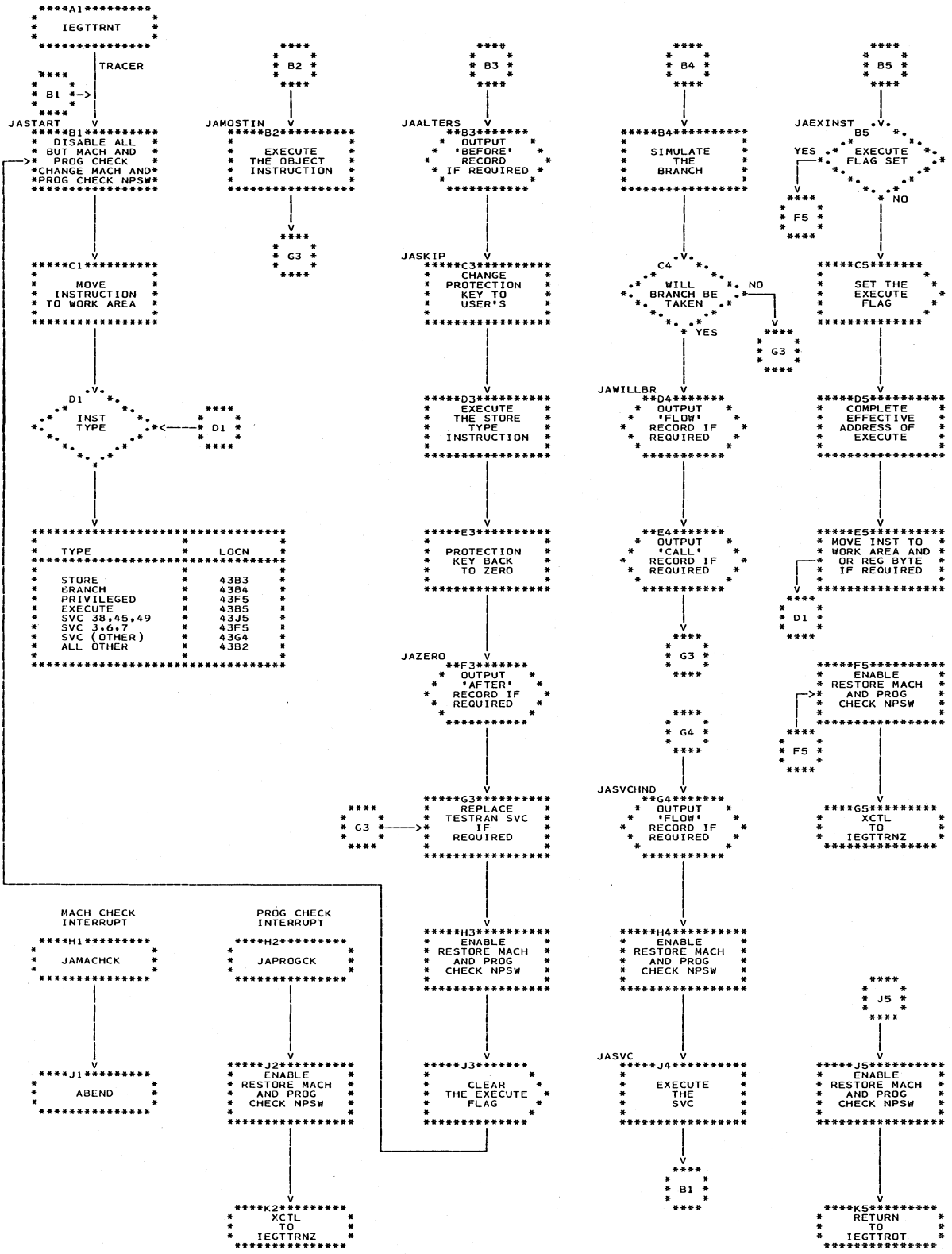


Chart 50. TESTRAN Editor

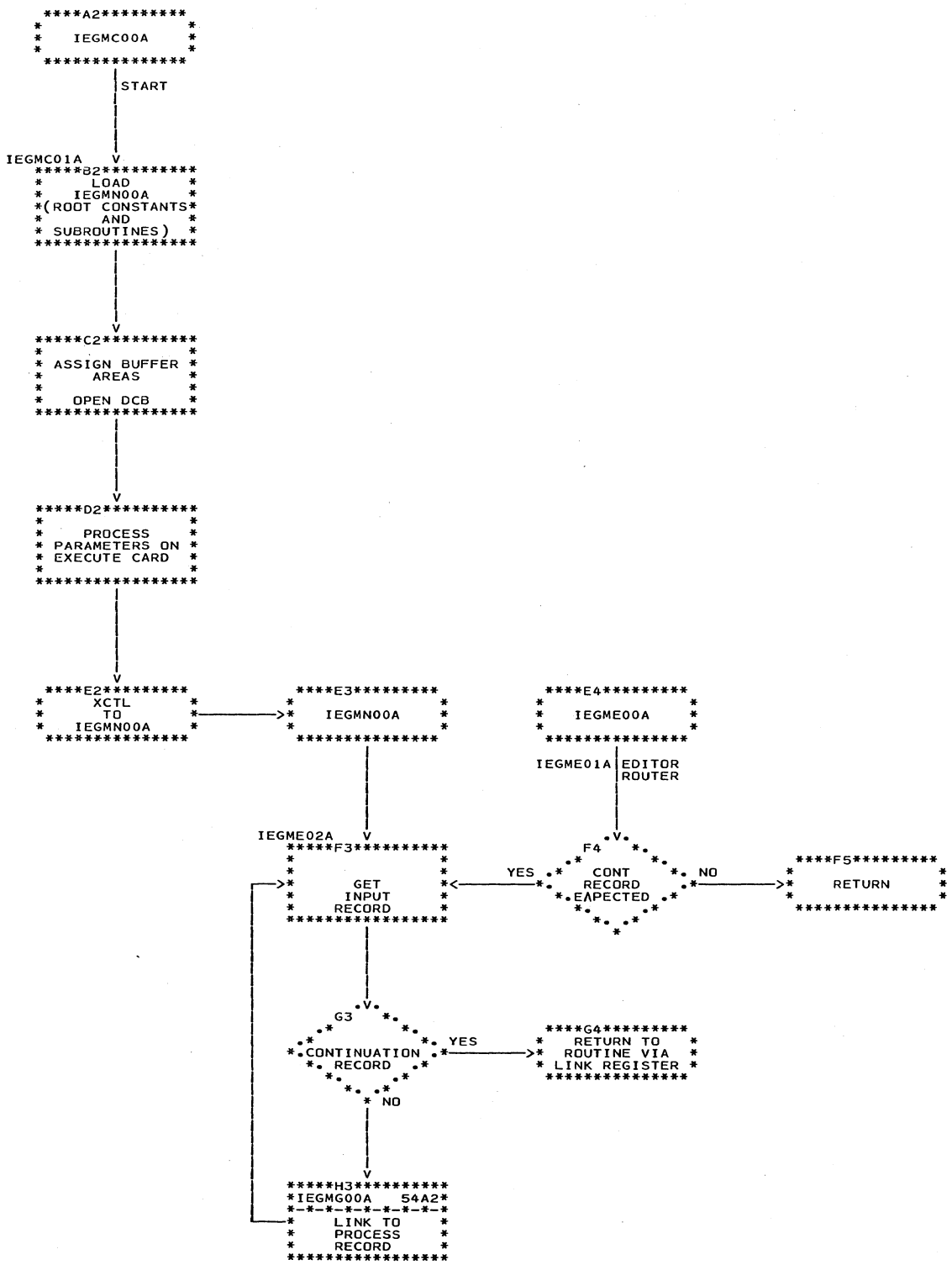
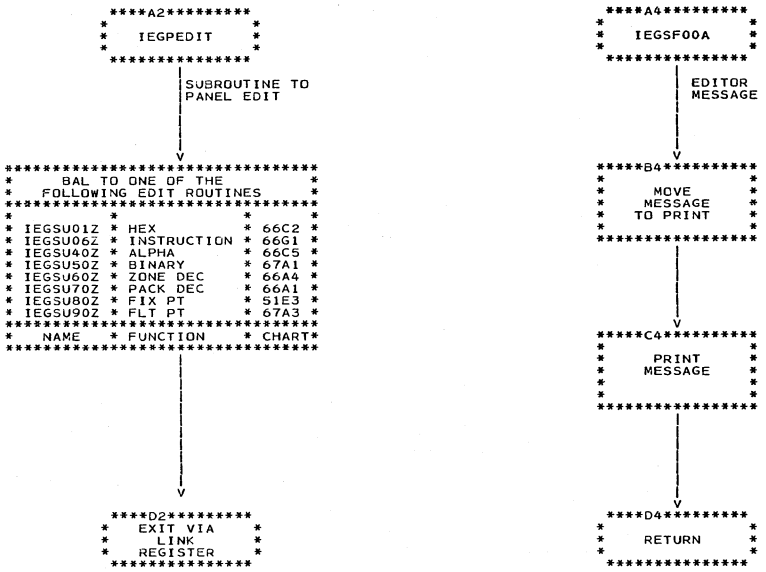


Chart 51. TESTRAN Editor



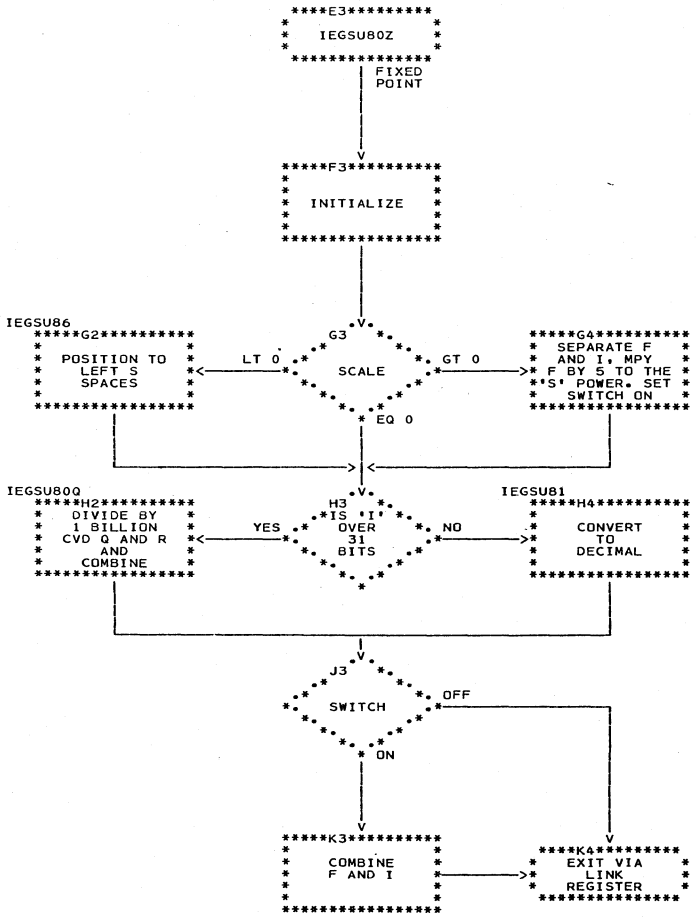


Chart 52. TESTRAN Editor

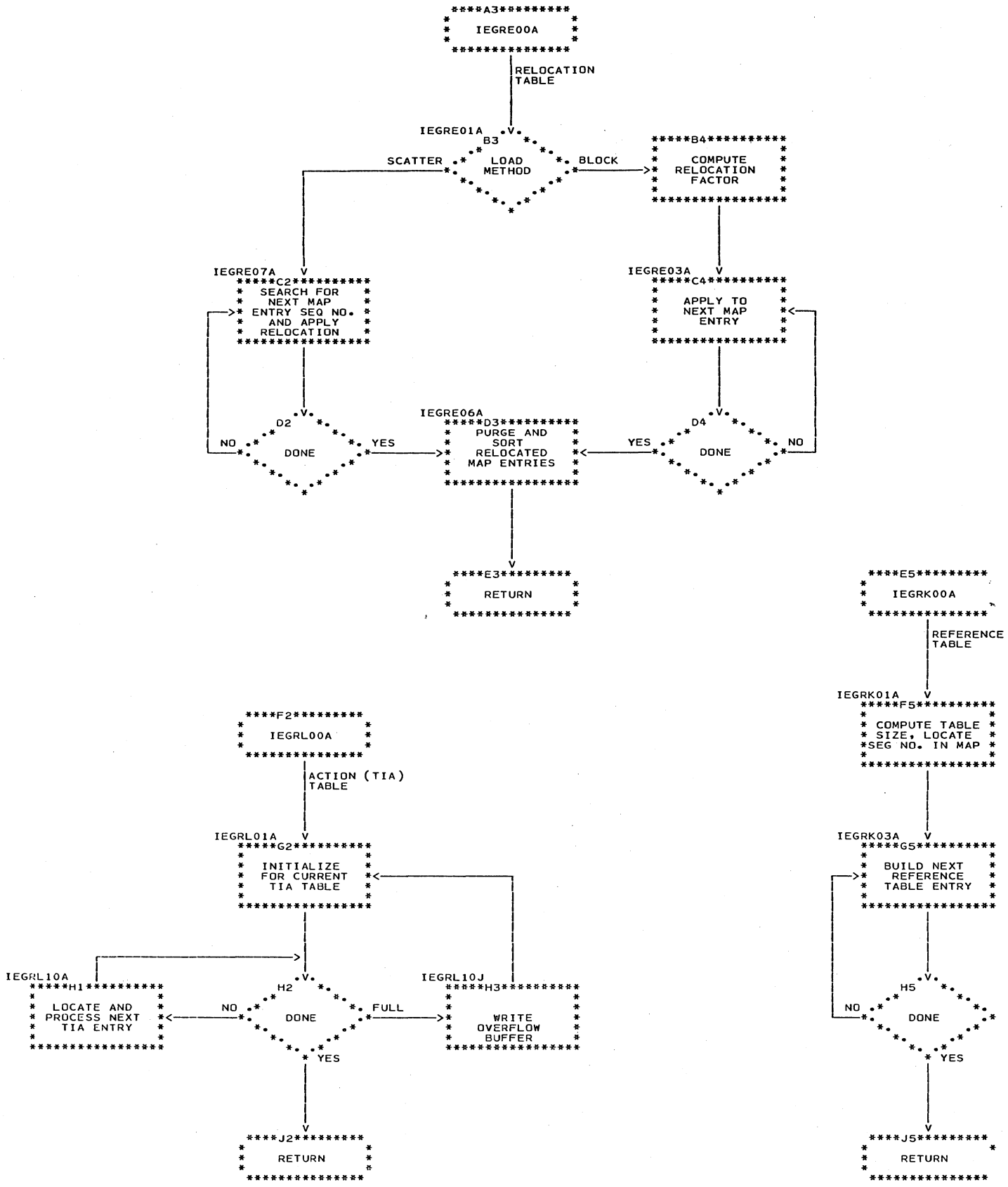


Chart 53. TESTRAN Editor

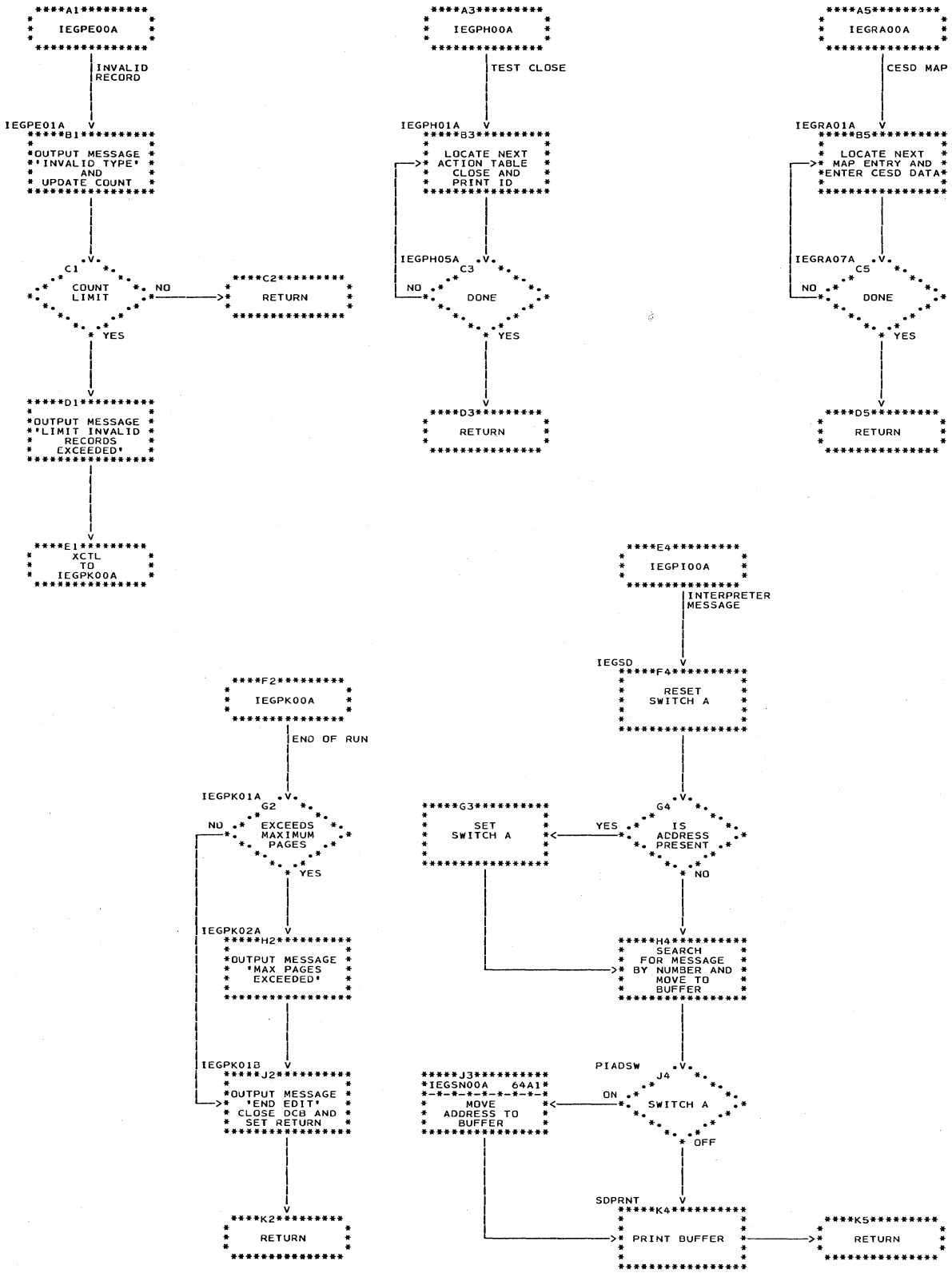


Chart 54. TESTRAN Editor

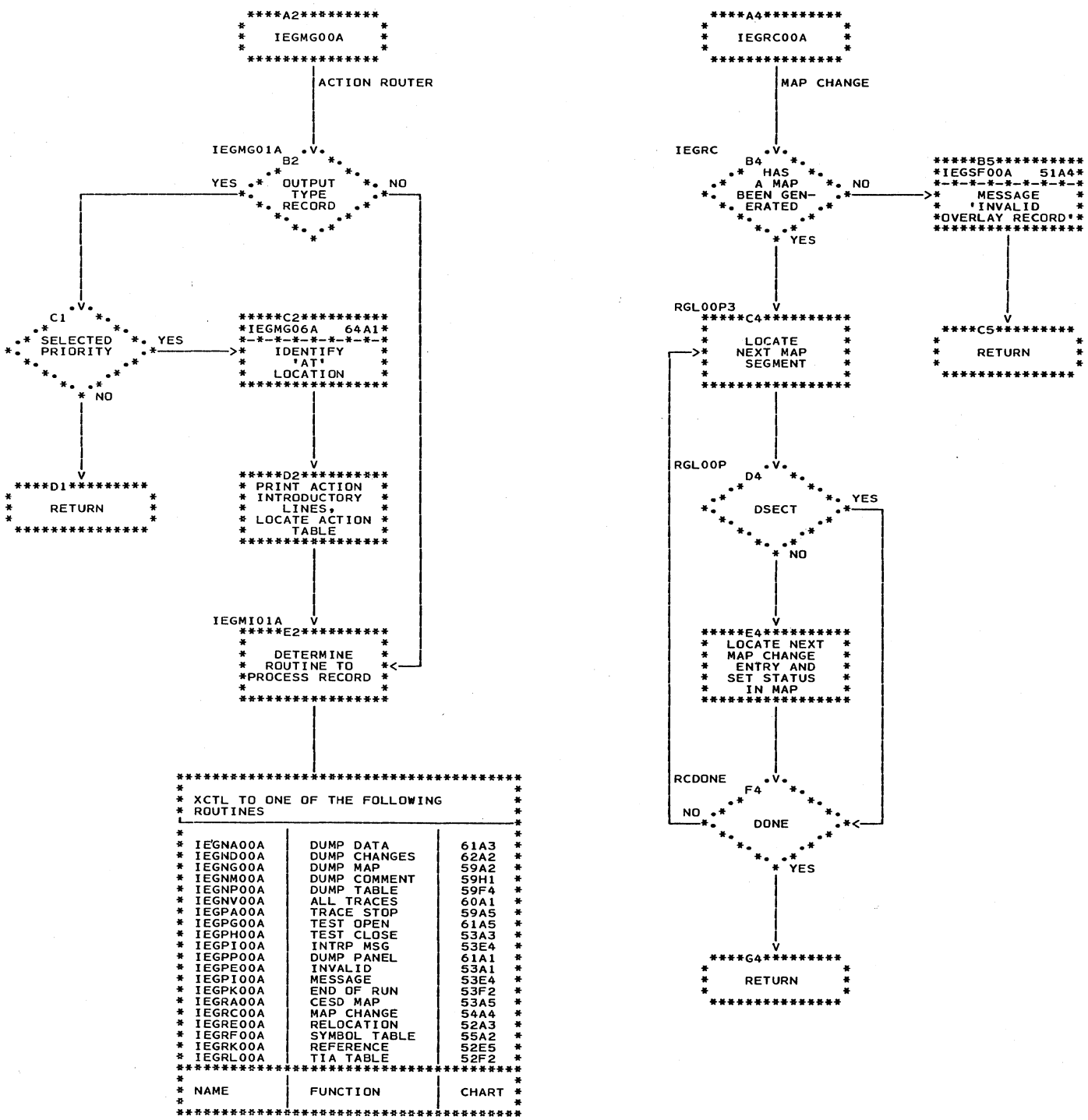


Chart 55. TESTRAN Editor

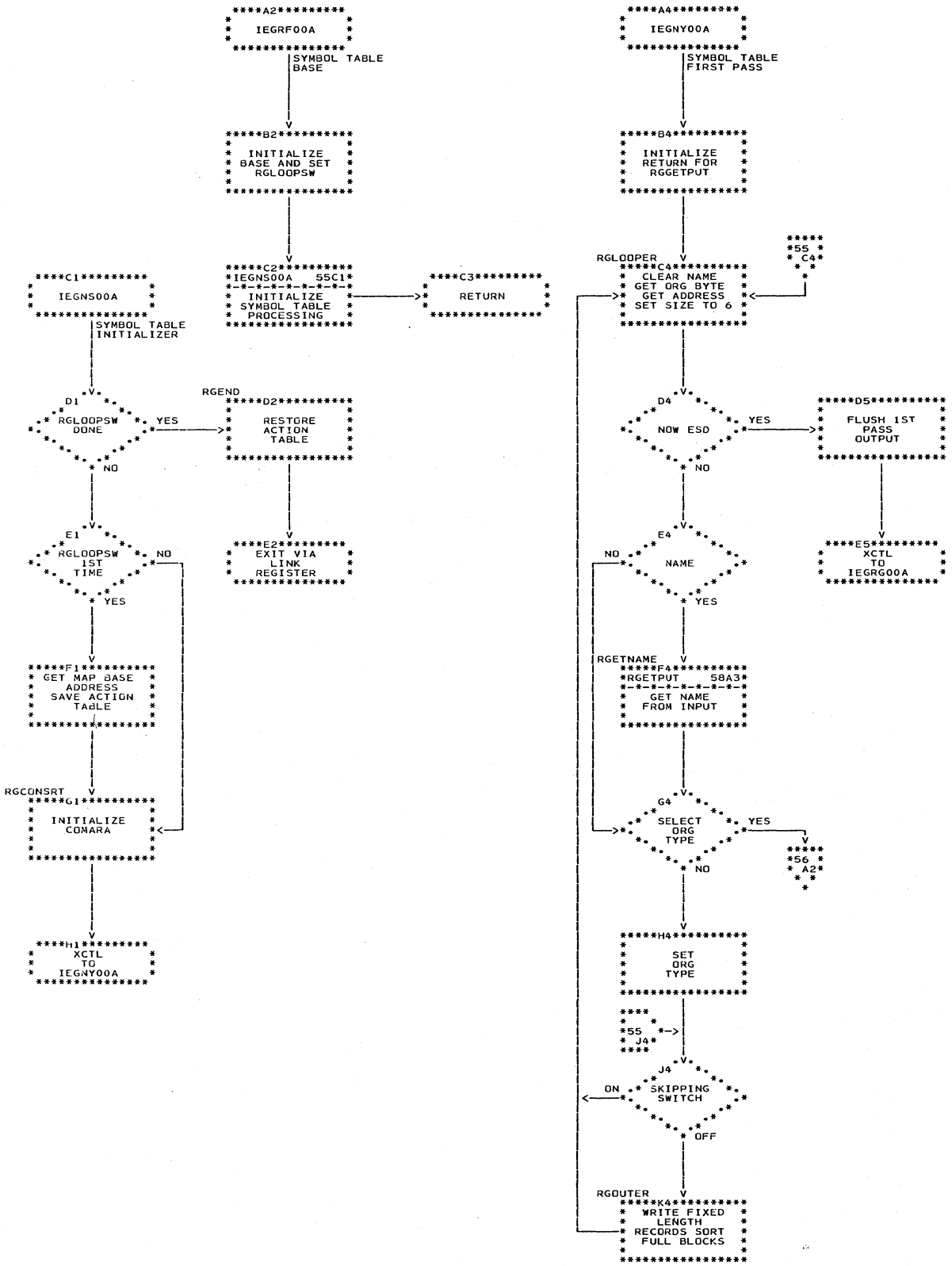


Chart 56. TESTRAN Editor

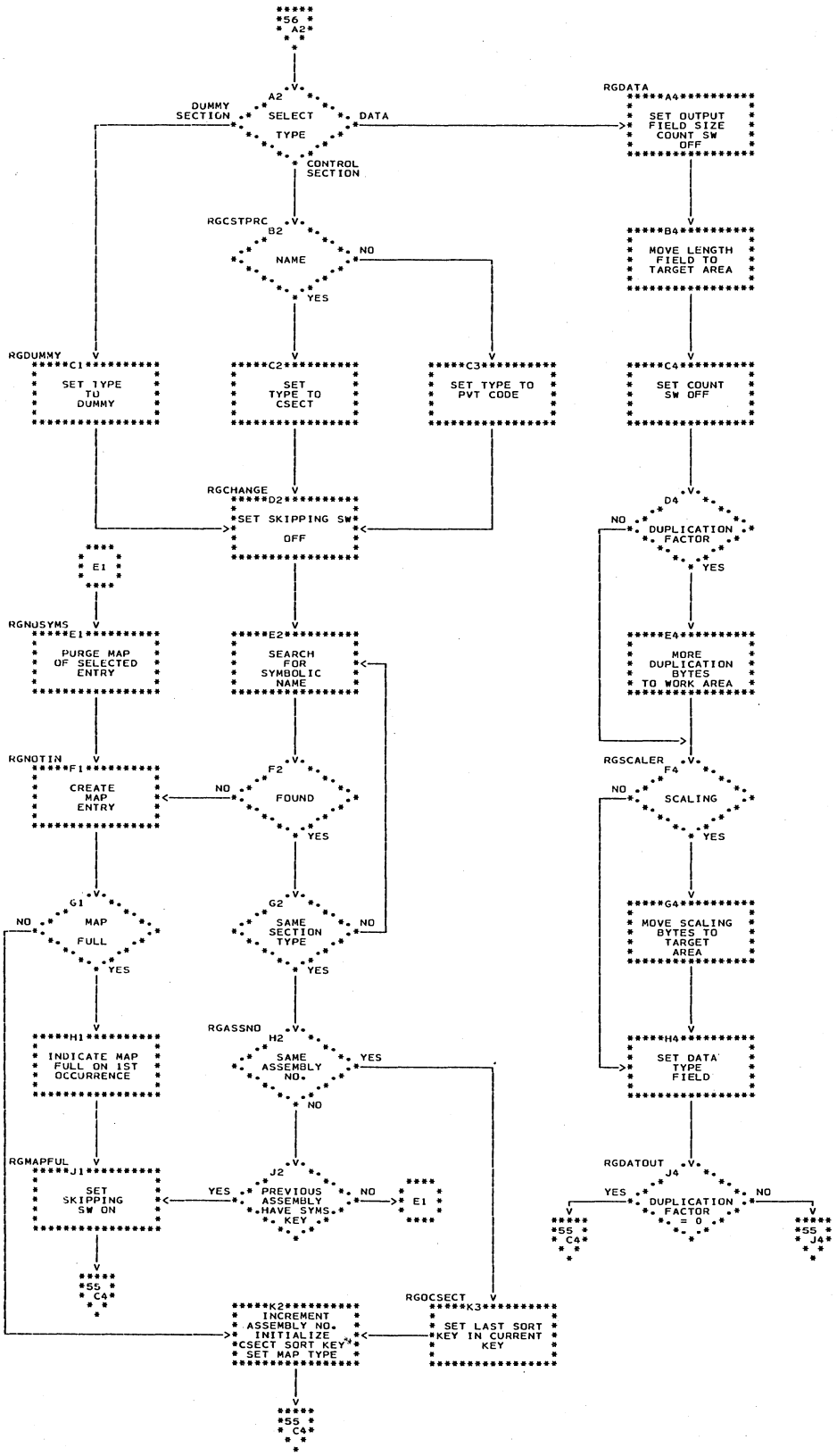


Chart 57. TESTRAN Editor

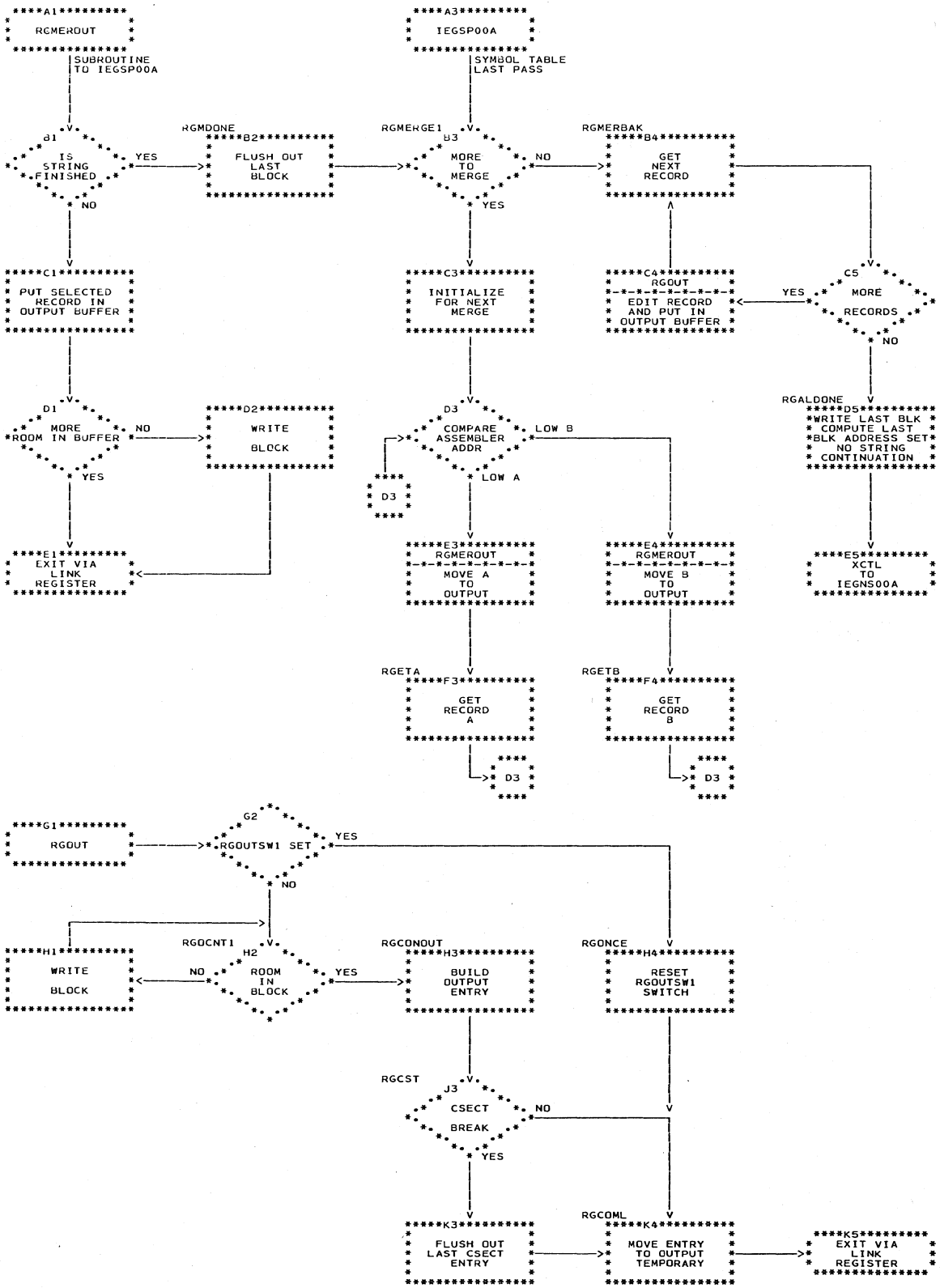


Chart 58. TESTRAN Editor

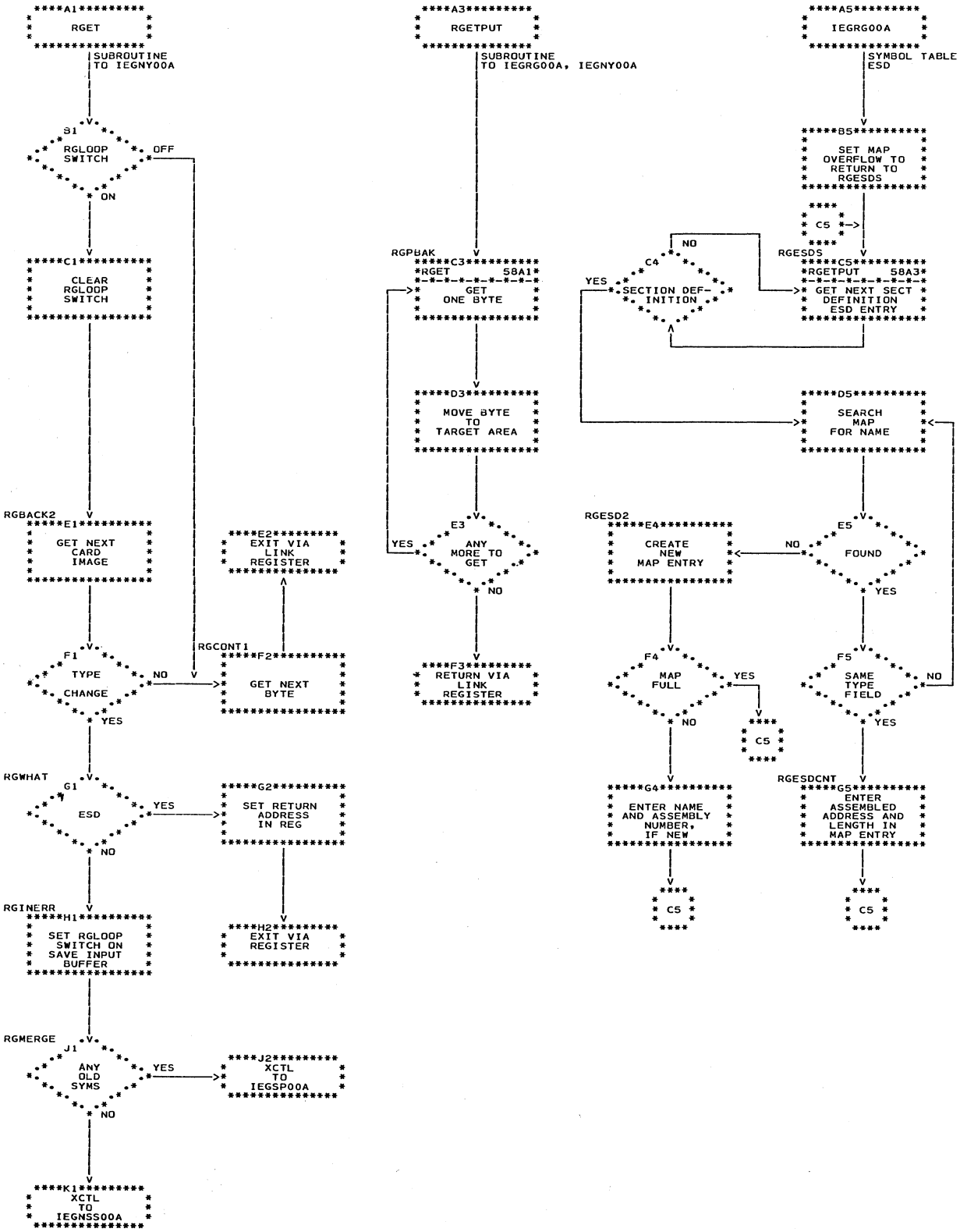


Chart 59. TESTRAN Editor

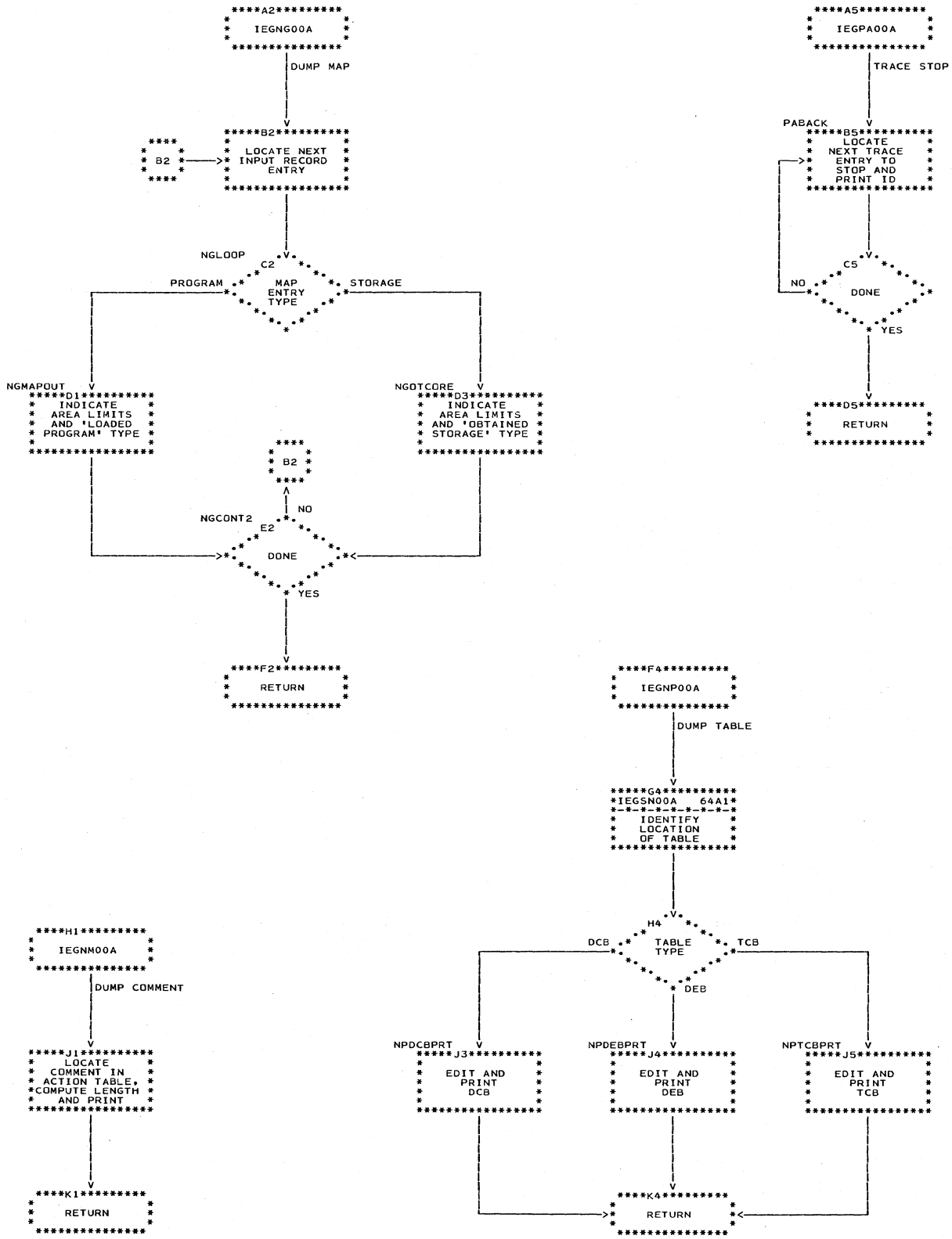


Chart 60. TESTRAN Editor

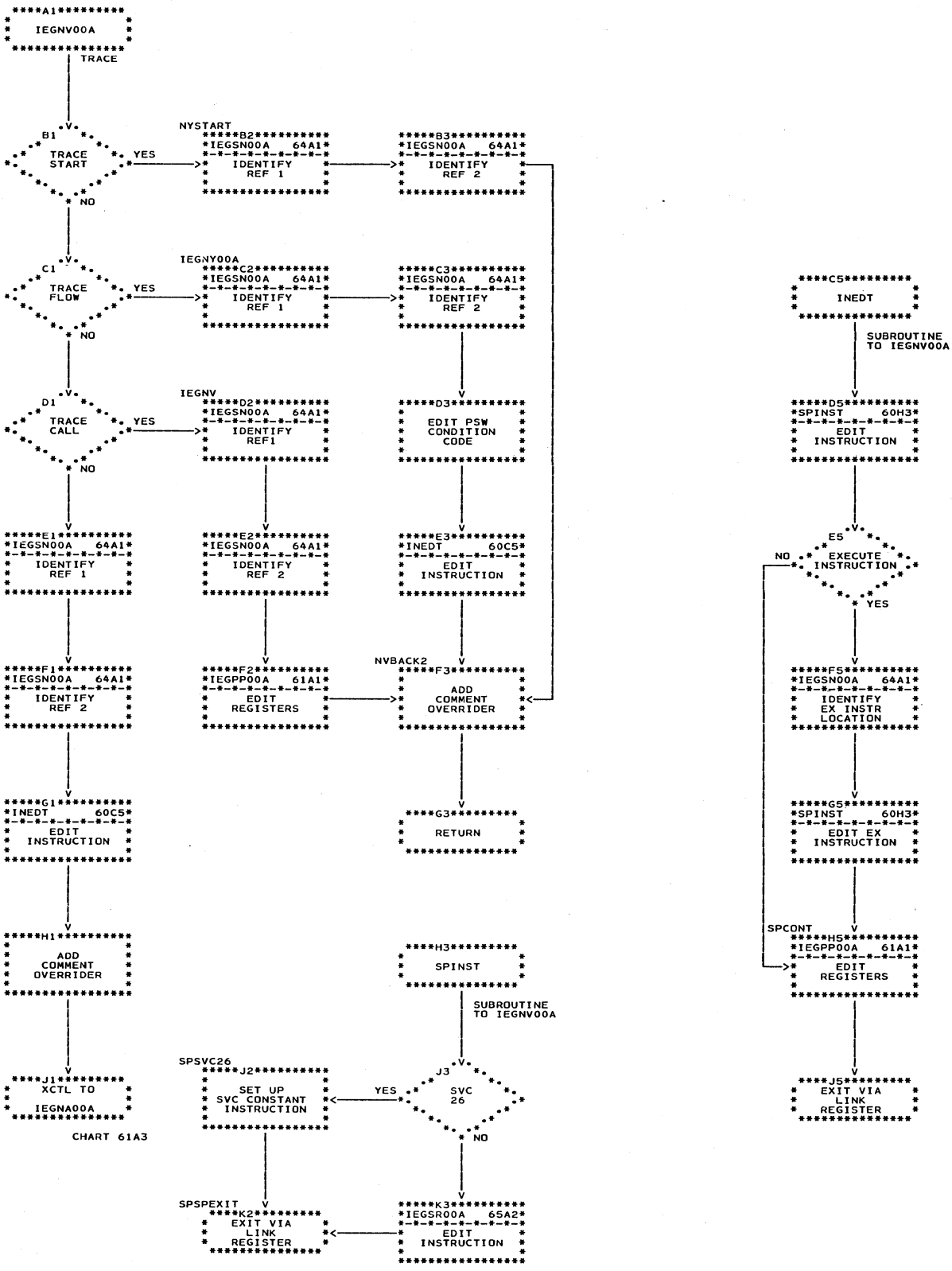


Chart 61. TESTRAN Editor

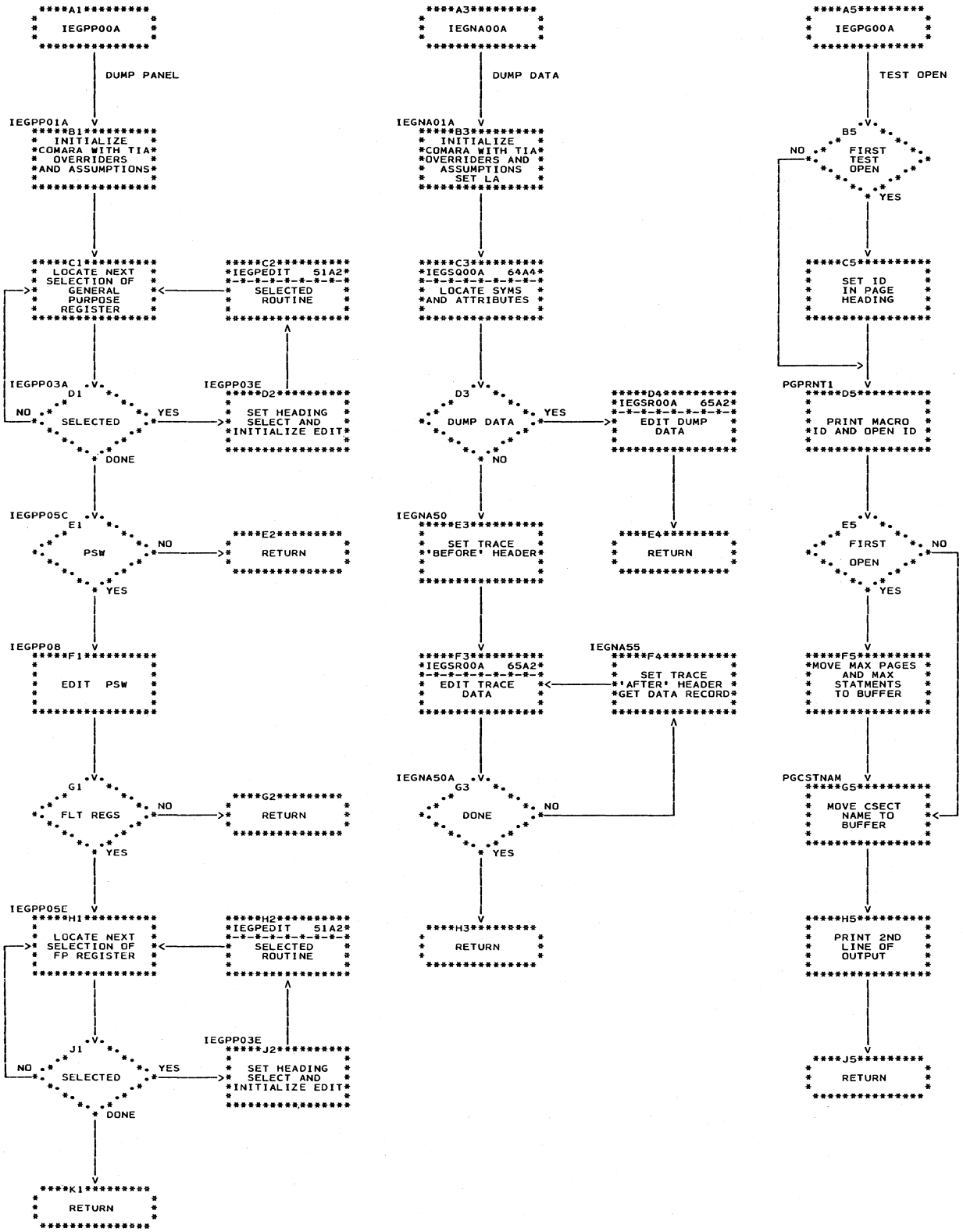


Chart 62. TESTRAN Editor

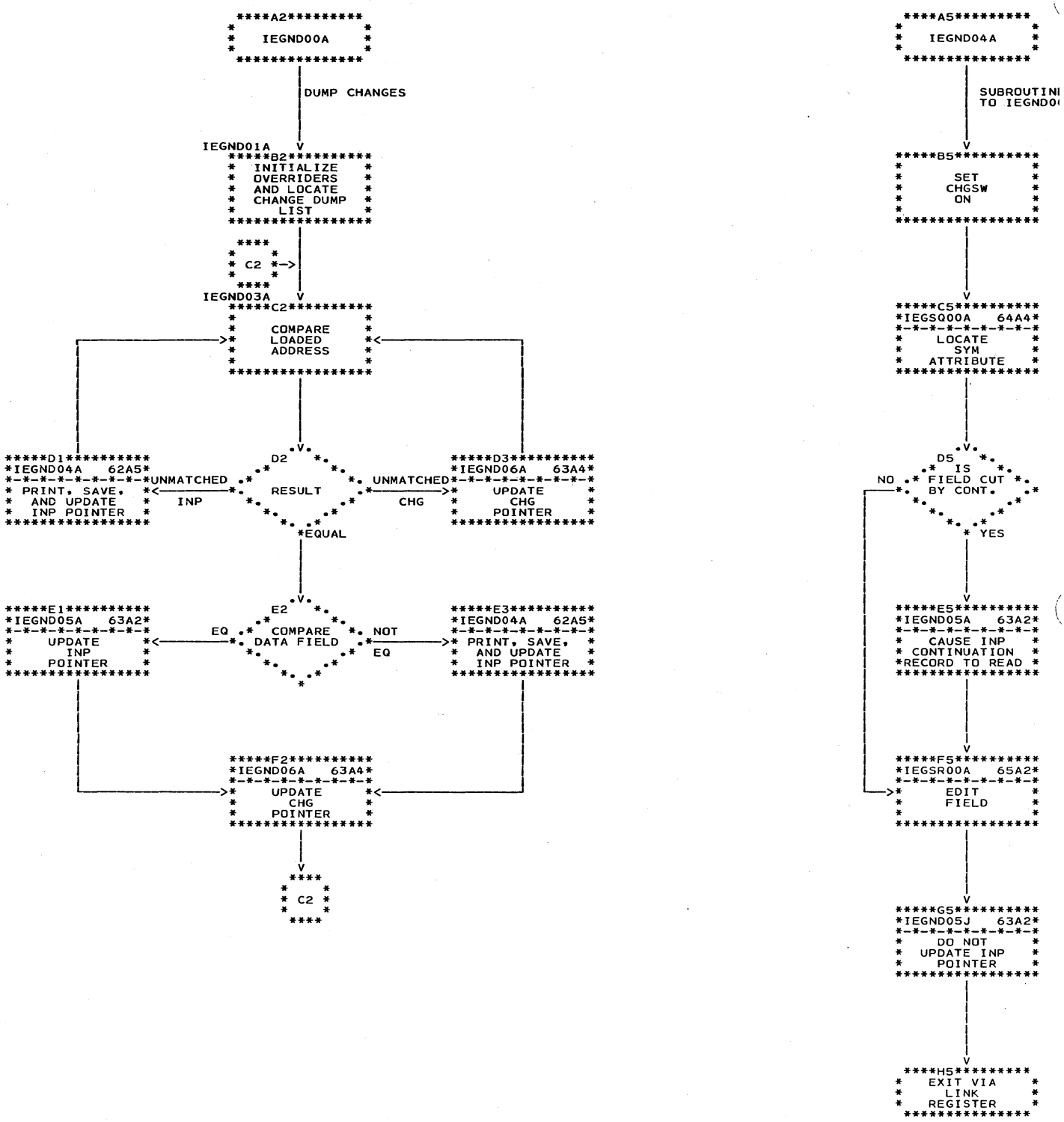


Chart 63. TESTRAN Editor

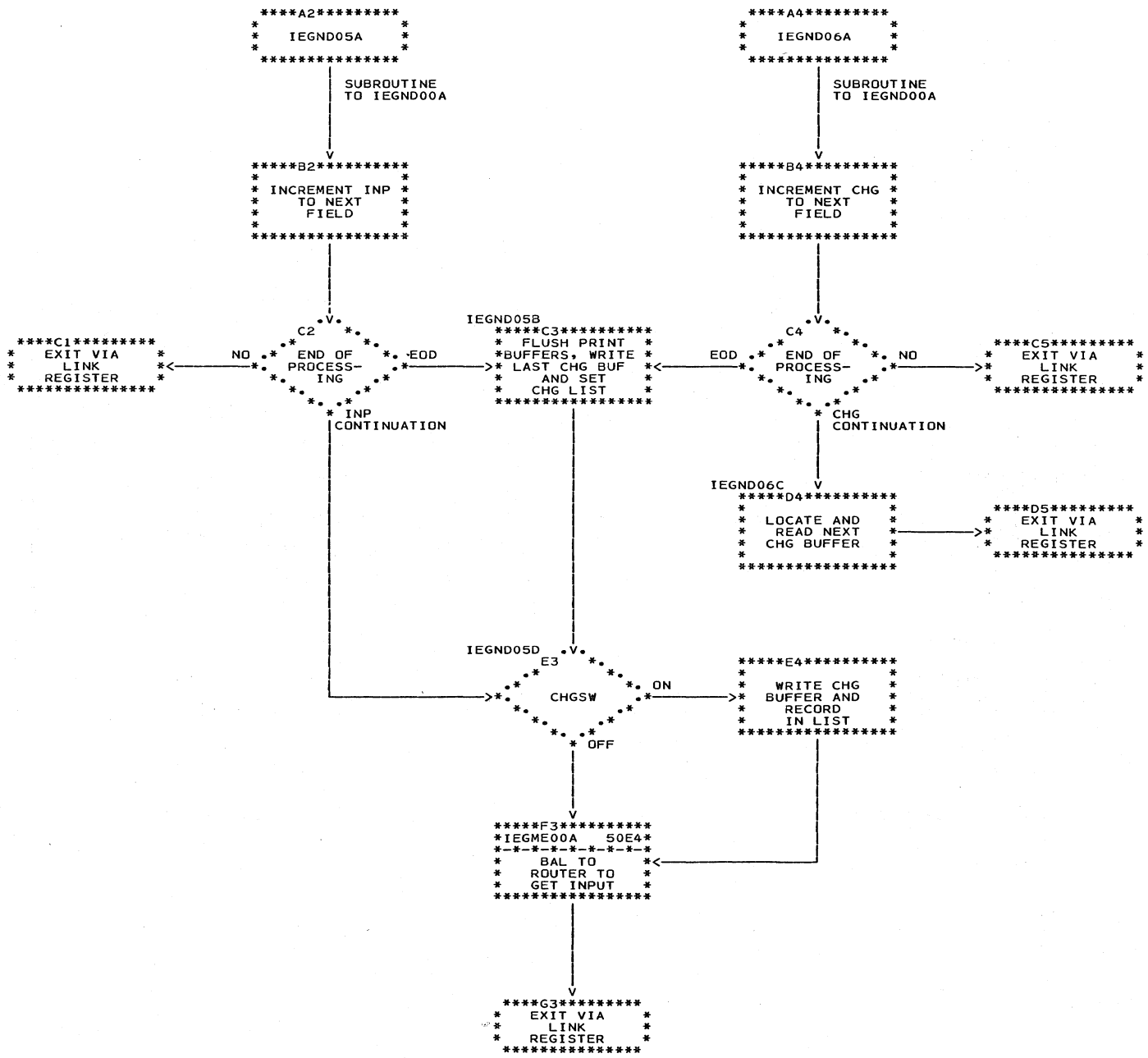


Chart 64. TESTRAN Editor

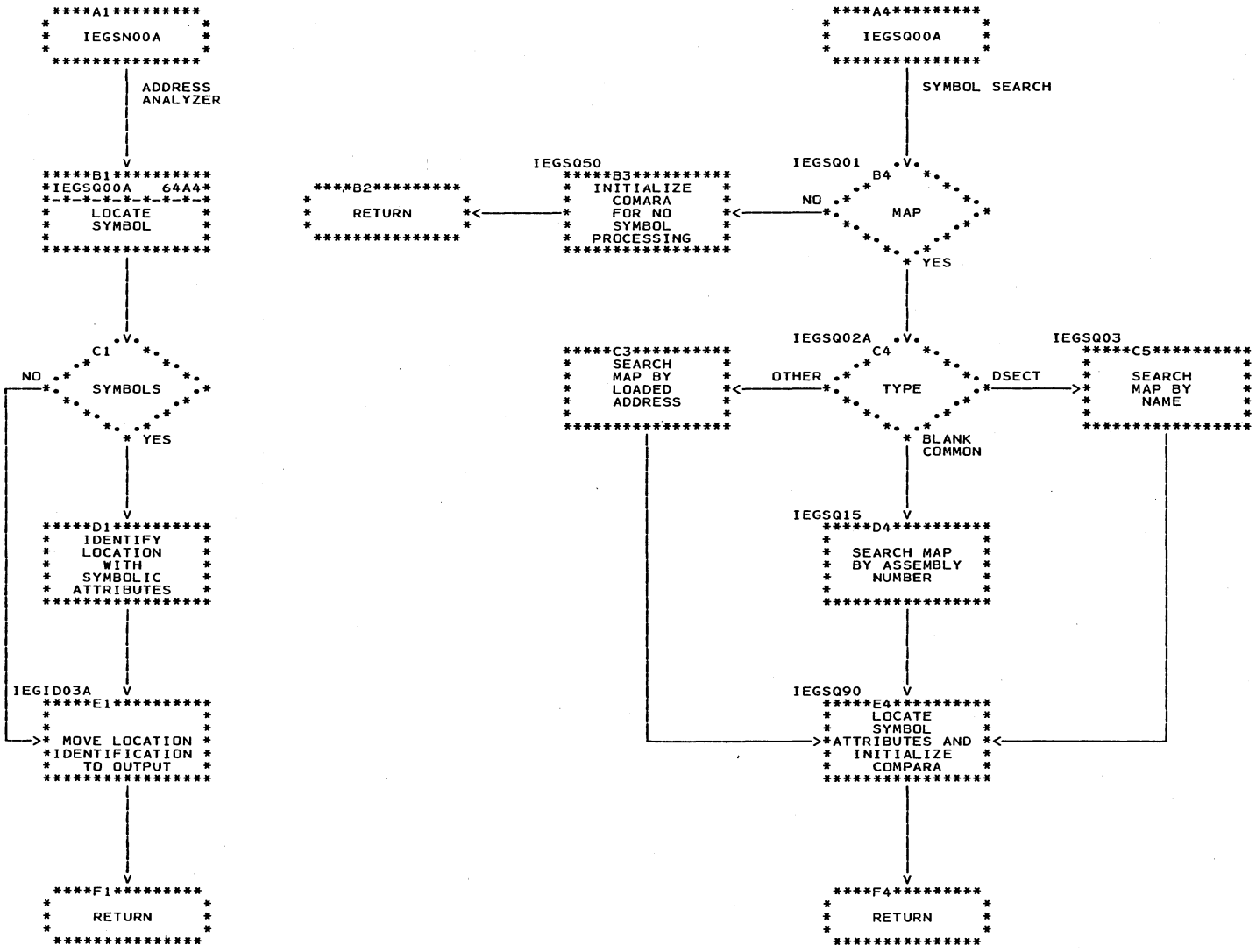


Chart 65. TESTRAN Editor

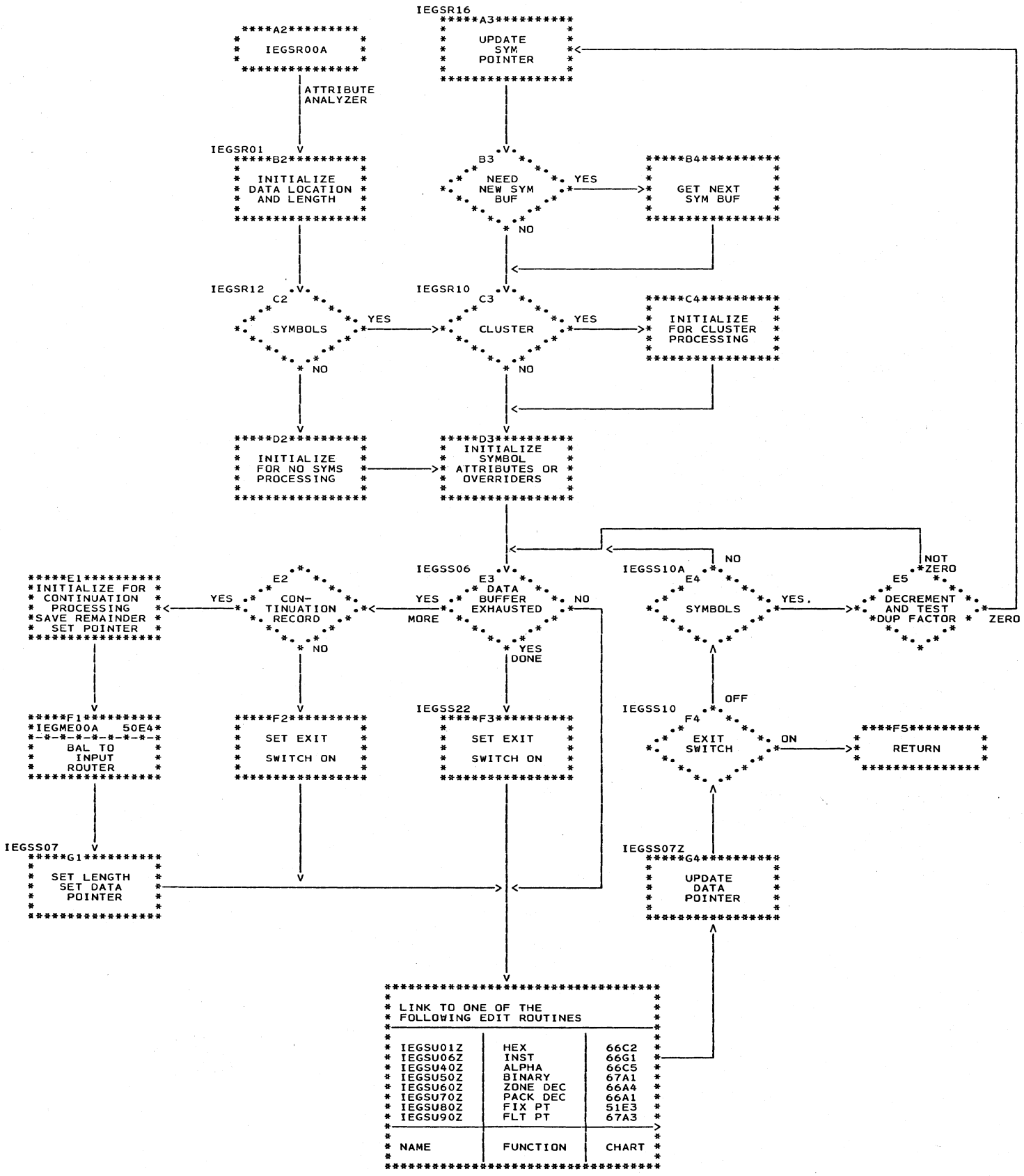


Chart 66. TESTRAN Editor

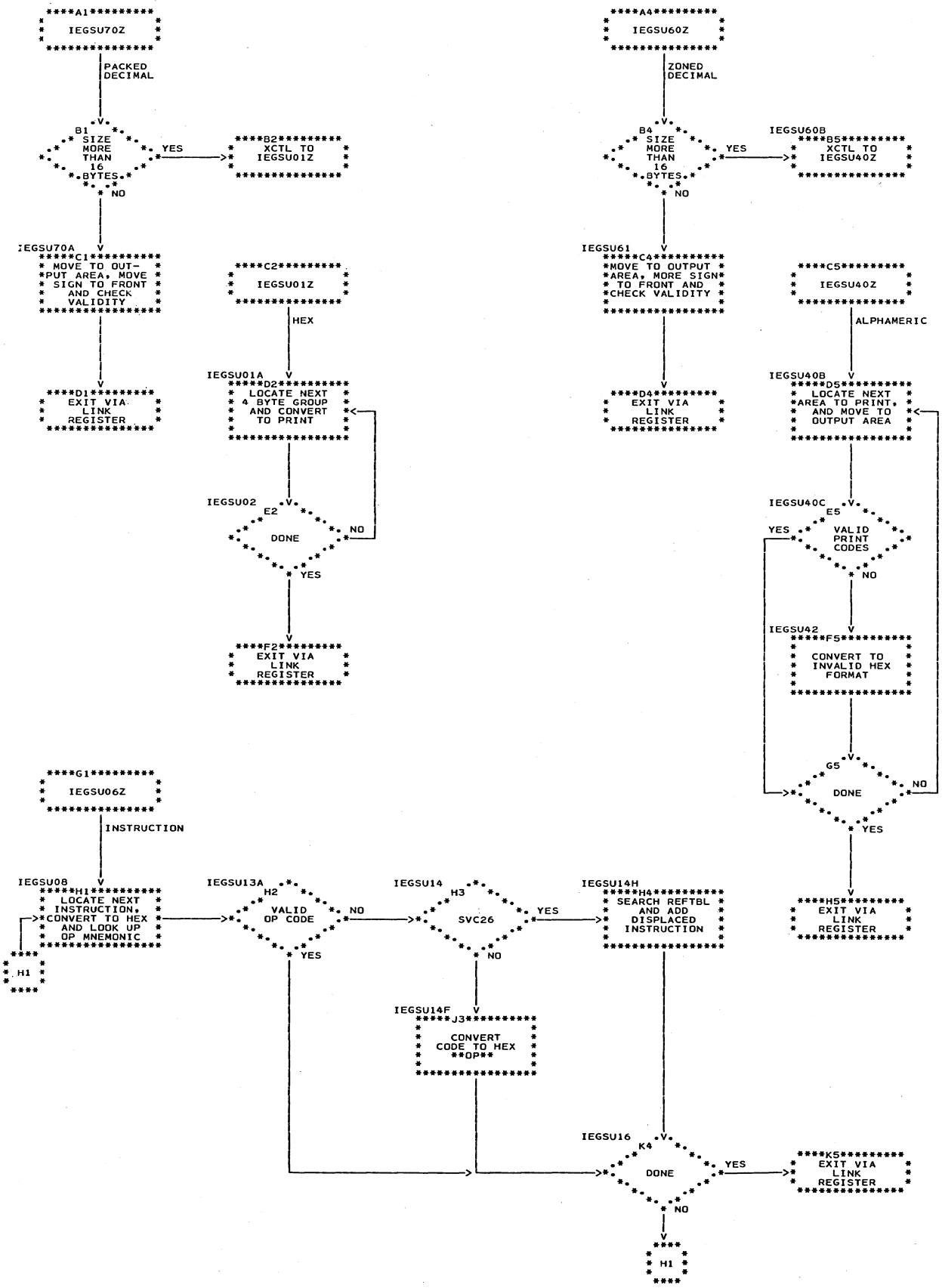
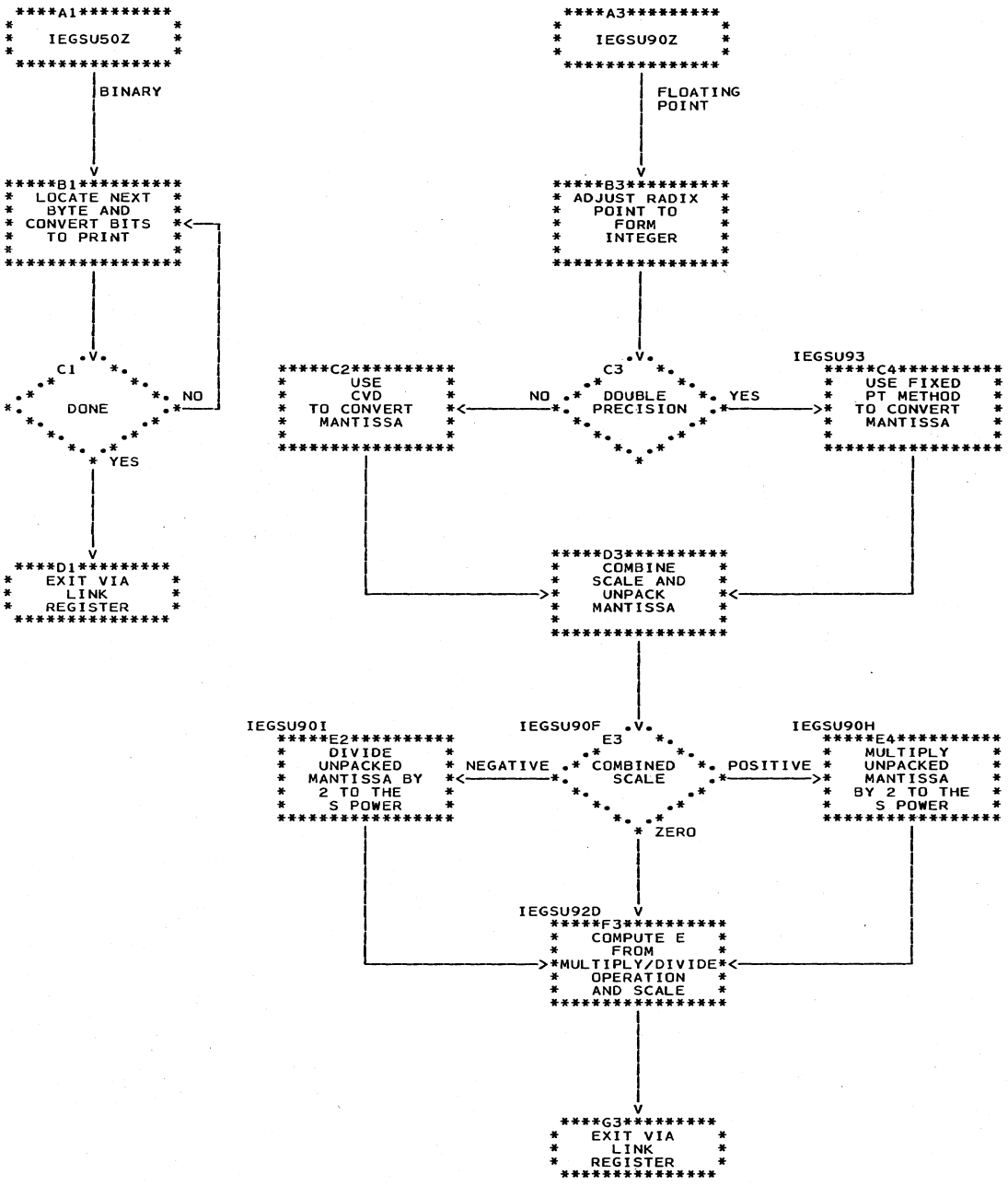


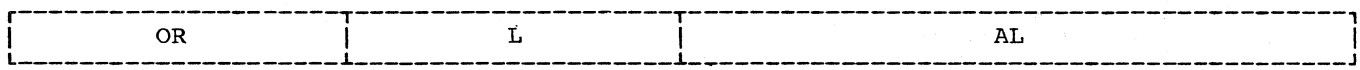
Chart 67. TESTRAN Editor



This appendix contains pictorial representations of each of the 23 TIA table entry types. The "A" (address) and modifier fields appear in many of the entries and are explained fully at the beginning of this appendix and merely named in the individual entry presentations.

A FIELD EXPANSION

The A fields included in the TIA entries are expanded as follows:



OR (1 byte) Organization Byte
- Bit 0 - "L" Field Flag
  - 1 = Present
  - 0 = Not present
- Bit 1 - Not Used
- Bits 2,3 - Type of "AL" Field present
  - 00 = Literal
  - 01 = 24 bit absolute address
  - 10 = 16 bit base displacement address
  - 11 = 1 byte register number
    The contents of this register are to be used as a value rather than an address. This type will only appear in the SET family of macros and in the TEST WHEN and TEST ON macros.
- Bits 4-7 - Index Register Number
  - 0 = No indexing
  - 1-15 = The number of the general register to be used for indexing the address contained in the "AL" field.

L (1 byte) Length
If the "AL" field contains a literal, this field contains the length of that literal.

AL (1-255 bytes)
This field will vary depending on bits 2 and 3 of the organization byte as follows:

- 00 Literal - Length will be specified in the"L" field
- 01 24 bit Absolute Address
- 10 16 Bit Base/Displacement Address
  - Bits 0-3 - Base register
  - Bits 4-15 - Displacement
- 11 1 Byte Register Number
  - Bit 0
    - 1 = Floating point
    - 0 = General
  - Bits 1-3 - Not Used
  - Bits 4-7 - Register Number

MODIFIER FIELDS S, P, FO, L

In the variable field descriptions on subsequent pages, modifier fields are defined as follows:

S (2 bytes) Scale Modifier
A two byte signed integer field which specifies the number of places the decimal or binary point is to be moved to the left.

P (1-byte) Output Selection Code
A one byte field containing a hexadecimal coded value of 1-8 used by the TESTRAN Editor to determine the processing of test data.

80 Output Selection Code 1
40 Output Selection Code 2
20 Output Selection Code 3
10 Output Selection Code 4
8 Output Selection Code 5
4 Output Selection Code 6
2 Output Selection Code 7
1 Output Selection Code 8

FO (1 byte) Format Modifier

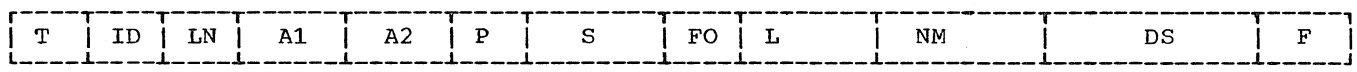
00 Character
01 Hexadecimal
02 Fixed Point[1]
03 Floating Point[1]
04 Packed Decimal
05 Zoned Decimal
0A Binary
0B Instruction

L (2-byte) Length Modifier
Overrides any previously defined length attributes for the data referred to by the macro-instruction.

---

[1]Length will always be present to distinguish between half-word and full-word fixed point and between short and long floating point.

DUMP DATA ENTRIES



T (1 byte) Type Entry
06 for DUMP DATA

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
The length of this entry in bytes

A1,A2
These fields contain the from and to addresses.

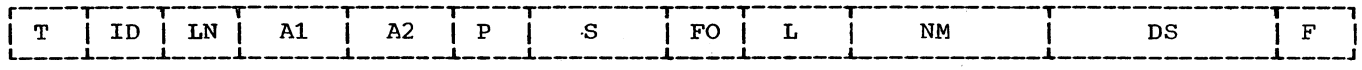
P,S,FO,L (S,L=2 bytes; P,FO,=1 byte each)
These fields specify an output selection code for output generated by this statement and overriders for attributes contained in the symbol table. If no symbol table exists, these are attribute specifications.

NM (2-9 bytes) Name Modifier
Byte 1 - Length of Symbolic Name
Bytes 2-9 - Symbolic Name

DS (5-12 bytes) Dummy Section Name
This field consists of 5-12 bytes of data as follows:
Byte 1 - Length (1-8) of the symbolic dummy section name
Byte 2 - Repeat Count (1-255)
Bytes 3,4 - Base/displacement address of dummy section
Bytes 5-12 - Symbolic name of dummy section

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bit 1 - "S" Field Flag
1 = Present
0 = Not present
Bit 2 - "FO" Field Flag
1 = Present
0 = Not present
Bit 3 - "L" Field Flag
1 = Present
0 = Not present
Bit 4 - "NM" Field Flag
1 = Present
0 = Not present
Bit 5 - "DS" Field Flag
1 = Present
0 = Not present
Bits 6,7 - Not Used

DUMP CHANGES ENTRY



T (1 byte) Type Entry
0E for DUMP CHANGES

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
The length of this entry in bytes

A1,A2
These fields contain the from and to addresses.

P,S,FO,L (S,L=2 bytes; P,FO,=1 byte each)
These fields specify an output selection code for output generated by this statement and overriders for attributes contained in the symbol table. If no symbol table exists, these are attribute specifications.

NM (2-9 bytes) Name Modifier
Byte 1 - Length of Symbolic Name
Bytes 2-9 - Symbolic Name

DS (5-12 bytes) Dummy Section Name
This field consists of 5-12 bytes of data as follows:
Byte 1 - Length (1-8) of the symbolic dummy section name
Byte 2 - Repeat Count (1-255)
Bytes 3,4 - Base/displacement address of dummy section
Bytes 5,12 - Symbolic name of dummy section

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bit 1 - "S" Field Flag
1 = Present
0 = Not present
Bit 2 - "FO" Field Flag
1 = Present
0 = Not present
Bit 3 - "L" Field Flag
1 = Present
0 = Not present
Bit 4 - "NM" Field Flag
1 = Present
0 = Not present
Bit 5 - "DS" Field Flag
1 = Present
0 = Not present
Bits 6,7 - Not Used

DUMP MAP ENTRY

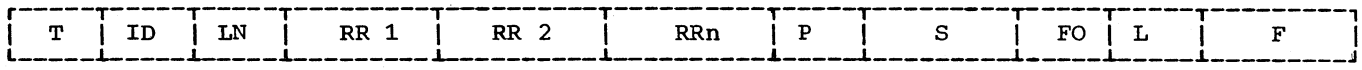
T (1 byte) Type Entry
12 for DUMP MAP

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

P (1 byte) Output Selection Code Specification

DUMP PANEL ENTRY



T (1 byte) Type Entry
16 for DUMP PANEL

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

RR1-RRn (1 or 2 bytes each)
Register request. Length of each of these entries depends on the first four bits as follows:

Bits 0-3 = 1
Individual general register request. Length of entry is one byte. Bits 4-7 specify general register to be dumped.

Bits 0-3 = 2
Individual floating point register request. Length of entry is one byte. Bits 4-7 specify floating point register to be dumped.

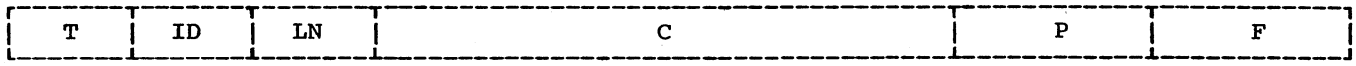
Bits 0-3 = 4
Range of general registers request. Length of entry is two bytes.
Bits 4-7 - First general register of range
Bits 8-11 - Not used
Bits 12-15 - Last general register of range

Bits 0-3 = 8
Range of floating point register request
Bits 4-7 - First floating point register of range
Bits 8-11 - Not used
Bits 12-15 - Last floating point register of range

P,S,FO,L (S,L=2 bytes; P,FO=1 byte each)
These fields specify an output selection code for output generated by this statement and overriders for attributes contained in the symbol table. If no symbol table exists, these are attribute specifications.

F (2 bytes) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bit 1 - "S" Field Flag
1 = Present
0 = Not present
Bit 2 - "FO" Field Flag
1 = Present
0 = Not present
Bit 3 - "L" Field Flag
1 = Present
0 = Not present
Bit 4-7 - Not Used
Bits 8-15 - Count of "RR" Fields

DUMMY COMMENT ENTRY



T (1 byte) Type Entry
1A for DUMP COMMENT

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

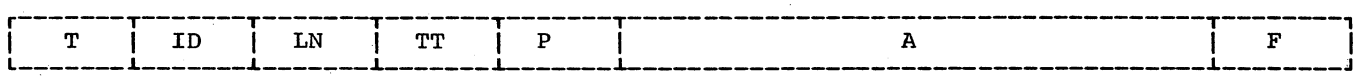
LN (1 byte) Length
Length of this entry in bytes

C (variable) Comment
Alphanumeric comment as supplied by the user

P (1 byte) Output Selection Code Specification

F (1 byte) Flag Byte
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1-7 - Length in Bytes of "C" Field.

DUMP TABLE ENTRY



T (1 byte) Type Entry
1E for DUMP TABLE

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

TT (1 byte) Table Type
This byte specifies the type of table to be dumped as per the following:
04 = DCB (Data Control Block)
08 = DEB (Data Extent Block)
0C = TCB (Task Control Block)

A (0-4 bytes) Name Address Field
This field varies depending on the table type contained in the 'TT' field. Contents of this field for each table type is as follows:
DCB Normal "A" field containing the DCB address
DEB Normal "A" field containing the DCB address
TCB This field is omitted for TCB

P (1 byte) Output Selection Code Specification

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1-6 - Not Used
Bit 7 - "A" Field Flag
1 = Present
0 = Not present

TRACE REFER ENTRY

T (1 byte) Type Entry
22 for TRACE REFER

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

A1,A2
These fields contain the from and to addresses.

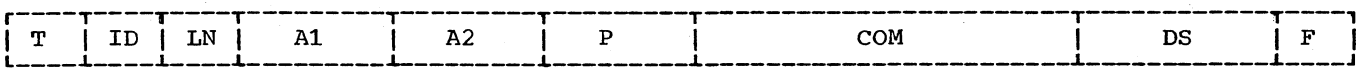
P,S,FO,L (S,L=2 bytes, P,FO=1 byte each)
These fields specify an output selection code for output generated by this statement and overriders for attributes contained in the symbol table. If no symbol table exists, these are attribute specifications.

COM (2-121 bytes) Comment Field
Byte 1 - Length of Comment
Bytes 2-121 - Alphanumeric Comment

DS (5-12 bytes) Dummy Section Name
This field consists of 5-12 bytes of data as follows:
Byte 1 - Length (1-8) of the symbolic dummy section name
Byte 2 - Repeat Count (1-255)
Bytes 3,4 - Base/displacement address of dummy section
Bytes 5-12 - Symbolic name of dummy section

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bit 1 - "S" Field Flag
1 = Present
0 = Not present
Bit 2 - "FO" Field Flag
1 = Present
0 = Not present
Bit 3 - "L" Field Flag
1 = Present
0 = Not present
Bit 4 - "COM" Field Flag
1 = Present
0 = Not present
Bit 5 - "DS" Field Flag
1 = Present
0 = Not present
Bits 6-7 - Not Used

TRACE CALL ENTRY



T (1 byte) Type Entry
26 for TRACE CALL

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

A1,A2
These fields contain the from and to addresses.

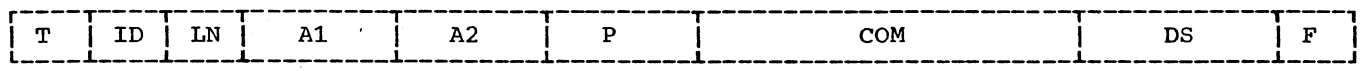
P (1 byte) Output Selection Code Specification

COM (2-121 bytes) Comment
Byte 1 - Length of Comment
Bytes 2-121 - Alphanumeric Comment

DS (5-12 bytes) Dummy Section Name
This field consists of 5-12 bytes of data as follows:
Byte 1 - Length (1-8) of the symbolic dummy section name
Byte 2 - Repeat Count (1-255)
Bytes 3,4 - Base/displacement address of dummy section
Bytes 5-12 - Symbolic name of dummy section

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1,2,3 - Not Used
Bit 4 - "COM" Field Flag
1 = Present
0 = Not present
Bit 5 - DS Field Flag
1 = Present
0 = Not present
Bit 6,7 - Not Used

TRACE FLOW ENTRY



T (1 byte) Type Entry
2A for TRACE FLOW

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

A1-A2
These fields contain the from and to addresses.

P (1 byte) Output Selection Code Specification

COM (2-121 bytes) Comment
Byte 1 - Length of Comment
Bytes 2-121 - Symbolic Comment

DS (5-12 bytes) Dummy Section Name
This field consists of 5-12 bytes of data as follows:
Byte 1 - Length (1-8) of the symbolic dummy section name
Byte 2 - Repeat Count (1-255)
Bytes 3,4 - Base/displacement address of dummy section
Bytes 5-12 - Symbolic name of dummy section

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1,2,3 - Not Used
Bit 4 - "COM" Field Flag
1 = Present
0 = Not present
Bit 5 - DS Field Flag
1 = Present
0 = Not present
Bits 6,7 - Not Used

TRACE STOP ENTRY

T (1 byte) Type Entry
2E for TRACE STOP

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

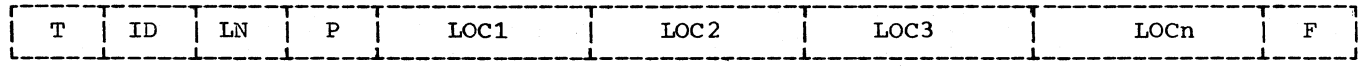
LN (1 byte) Length
Length of this entry in bytes

TA1,TAn (3 bytes each)
If no "TA" fields are present; all active trace macros will be stopped.
If any "TA" fields are present, they will contain addresses of specific macros whose action is to be stopped and only those listed will be stopped.

P (1 Byte) Output Selection Code Specification

F (1 Byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1-7 - Count of "TA" fields in this entry

TEST AT ENTRY



T (1 byte) Type Entry
02 for TEST AT

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

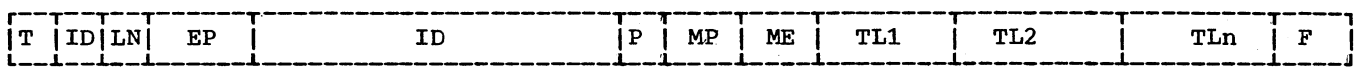
LN (1 byte) Length
Length of this entry in bytes

P (1 byte) Output Selection Code Specification

LOC1-LOCn (3 bytes each) AT Location
Each of these fields is a 24 bit absolute address at which the user has specified some TESTRAN action.

F (1 byte) Flags
Bit 0 - "P" Field Flag
1 = Present
0 = Not present
Bits 1-7 - Count of "A" field present

TEST OPEN ENTRY



T (1 byte) Type Entry
0A for TEST OPEN

ID (1 byte) Identification
For the TEST OPEN entry, the number of the SVC assigned to the TESTRAN open routine (49)

LN (1 byte) Length
Length of this entry in bytes

EP (3 bytes) Entry Point
This field contains the address to which TESTRAN is to give control after executing the TEST OPEN macro.

ID (8 bytes) Identification
This field contains an alphanumeric name, supplied by the user, to be used by TESTRAN to identify output of this table.

P (1 byte) Output Selection Code Specification

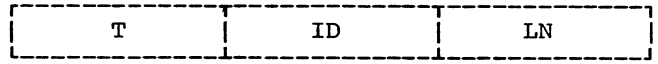
MP (2 bytes) Maximum number of Output pages the user desires TESTRAN to generate (1-65535)

ME (2 bytes) Maximum number of macros to be executed by the TESTRAN Interpreter

TL1-TLn (4 bytes each) Table Locations
Each of these fields contain the address of a secondary TEST OPEN macro to be loaded with this TEST OPEN

F (2 bytes) Flags
Bit 0 - "EP" Field Flag
1 = Present
0 = Not present
Bit 1 - Not used
Bit 2 - "ID" Field Flag
1 = Present
0 = Not present
Bit 3 - "P" Field Flag
1 = Present
0 = Not present
Bit 4 - "ML" Field Flag
1 = Present
0 = Not present
Bit 5 - "ME" Field Flag
1 = Present
0 = Not present
Bit 6 - Not used
Bit 7 - Transient or Non Transient mode flag as follows:
1 = LINK
0 = LOAD
This is a flag to inform the TESTRAN Router whether to link to or load required routines.
Bits 8-15 - TL count
These bits contain a count of the "TL" fields present in this TEST OPEN macro.

TEST CLOSE ENTRY



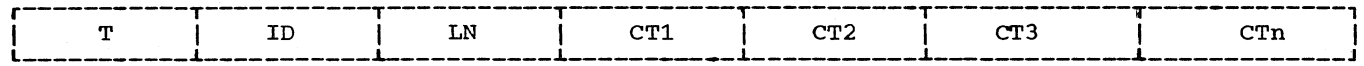
T (1 byte) Type Entry
32 for TEST CLOSE

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes.

Note: This macro does not have a variable field.

TEST DEFINE COUNTER ENTRY



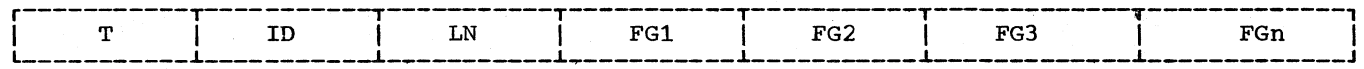
T (1 byte) Type Entry
42 for TEST DEFINE COUNTER

ID (1 BYTE) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

CT1-CTn (1 byte each)
Each of these one byte fields is a counter to be used by TESTRAN TEST ON macros. Their initial value is zero.

TEST DEFINE FLAG ENTRY



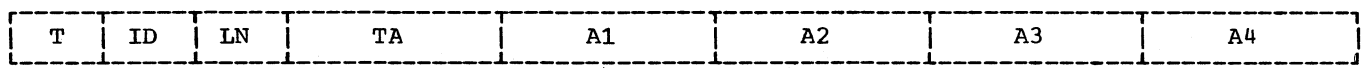
T (1 byte) Type Entry
46 for TEST DEFINE FLAG

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

FG1 - FGn (1 byte each)
Each of these are one byte fields which the user may use as a switch. Each flag has an initial value of zero (off).

TEST ON ENTRY



T (1 byte) Type Entry
56 for TEST ON

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes) TESTRAN Address
This field contains the 24 bit address of the next TESTRAN macro to be interpreted if a true conclusion is reached for this macro.

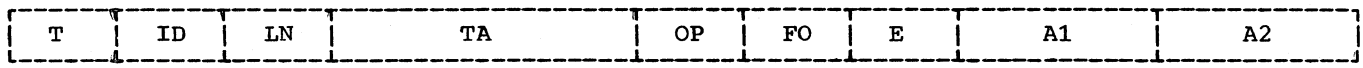
A1 Counter Field
This "A" field will contain either a one byte literal (initialized at zero) to be used as a counter or the 4 byte address of a counter defined with a test define counter macro.

A2-A4
These 3 "A" fields will each contain one of the items listed below.

1. Literal: A literal value to be used in determining a true or false conclusion for this macro.
2. Address: The address of a four byte variable, in the object program to be used in determining a true or false conclusion for this macro.
3. Register Number: A general register whose contents are to be used for determining a true or false conclusion.

A2 - Low Limit
A3 - High Limit
A4 - Interval

TEST WHEN ENTRY



T (1 byte) Type Entry
5A for TEST WHEN

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes) TESTRAN Address
This field contains the 24 bit address of the next TESTRAN macro to be interpreted if a true conclusion is reached for this macro.

OP (1 byte) Operator
This field contains an operator specifying the relation to be checked between A1 and A2 as follows:
00 = Equal
04 = Not Equal
08 = Less Than
0C = Greater Than
10 = Less than or equal to
14 = Greater than or equal to
18 = AND[1]
1C = OR[1]
20 = Independent Field (A2 Not Present)[1]

FO (1 byte) Data Format

E (1 byte) Extent
This field specifies the length of the data items to be compared.

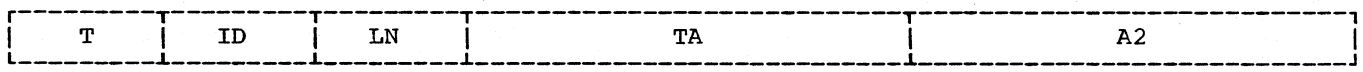
A1,A2 Address Fields
These fields will each contain one of the following.

1. Literal: A literal to be used as one item in the compare.
2. Address: The address of a variable or constant to be used in the compare.
3. Register Number: The number of a general or floating point register whose contents are to be used in the compare.

---

[1]No FO or L fields will exist.

SET COUNTER ENTRY



T (1 byte) Type Entry
4A for SET COUNTER

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

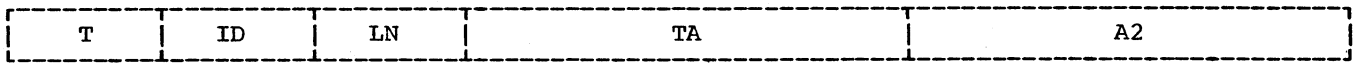
LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes) TIA Table Address
This field contains the address of the counter to be set.

A2
This field will contain one of the three items listed below.

1. Literal: The counter is to be set to the value of this literal.
2. Register Number: The counter is to be set to the value contained in this General Register.
3. Address: The counter is to be set to the value contained in the full word located at this address.

SET FLAG ENTRY



T (1 byte) Type Entry
4E for SET FLAG

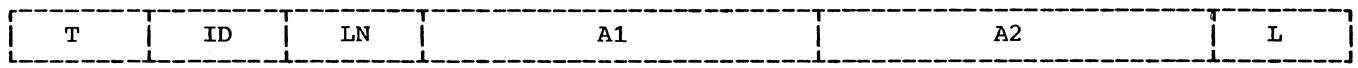
ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes)
This field will contain the address of the TESTRAN flag to be set. This flag must be defined elsewhere with a TEST DEFINE FLAG macro.

A2
This "A" field will contain either an integer (must be 0 or 1) or the address of another TESTRAN flag. If the field contains an integer, the flag will be set to that value. If the field contains another flag address, the flag specified in the 'TA' field will be set to the value of the flag at that address.

SET VARIABLE ENTRY



T (1 byte) Type Entry
52 for SET VARIABLE

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

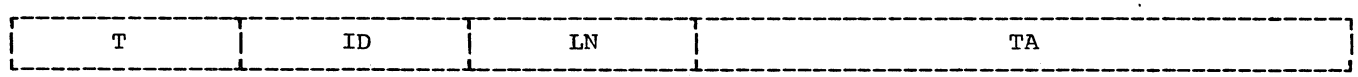
LN (1 byte) Length
Length of this entry in bytes

A1,A2 Address Fields
These fields may contain either of the two items listed below.

1. Address: An object program address.
2. Register Number: A general or floating point register number. In addition, the A2 field may contain a literal.

L (1 byte) Length of data fields specified

GO IN ENTRY



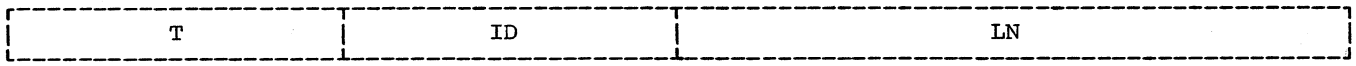
T (1 byte) Type Entry
36 for GO IN

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes) TIA Table Address
This field contains the address of the TIA Table entry to be interpreted next.

GO OUT ENTRY



T (1 byte) Type Entry
3A for GO OUT

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
Length of this entry in bytes

Note: This macro does not have a variable field.

GO BACK ENTRY

T (1 byte) Type Entry
3E for GO BACK

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

LN (1 byte) Length
The length of this entry in bytes

A1 Address Field
This field when present will contain an object program address to which the programmer wants TESTRAN to return control.

GO TO ENTRY

T (1 byte) Type Entry
5E for GO TO

ID (1 byte) Identification
The number assigned to this macro-instruction at expansion time

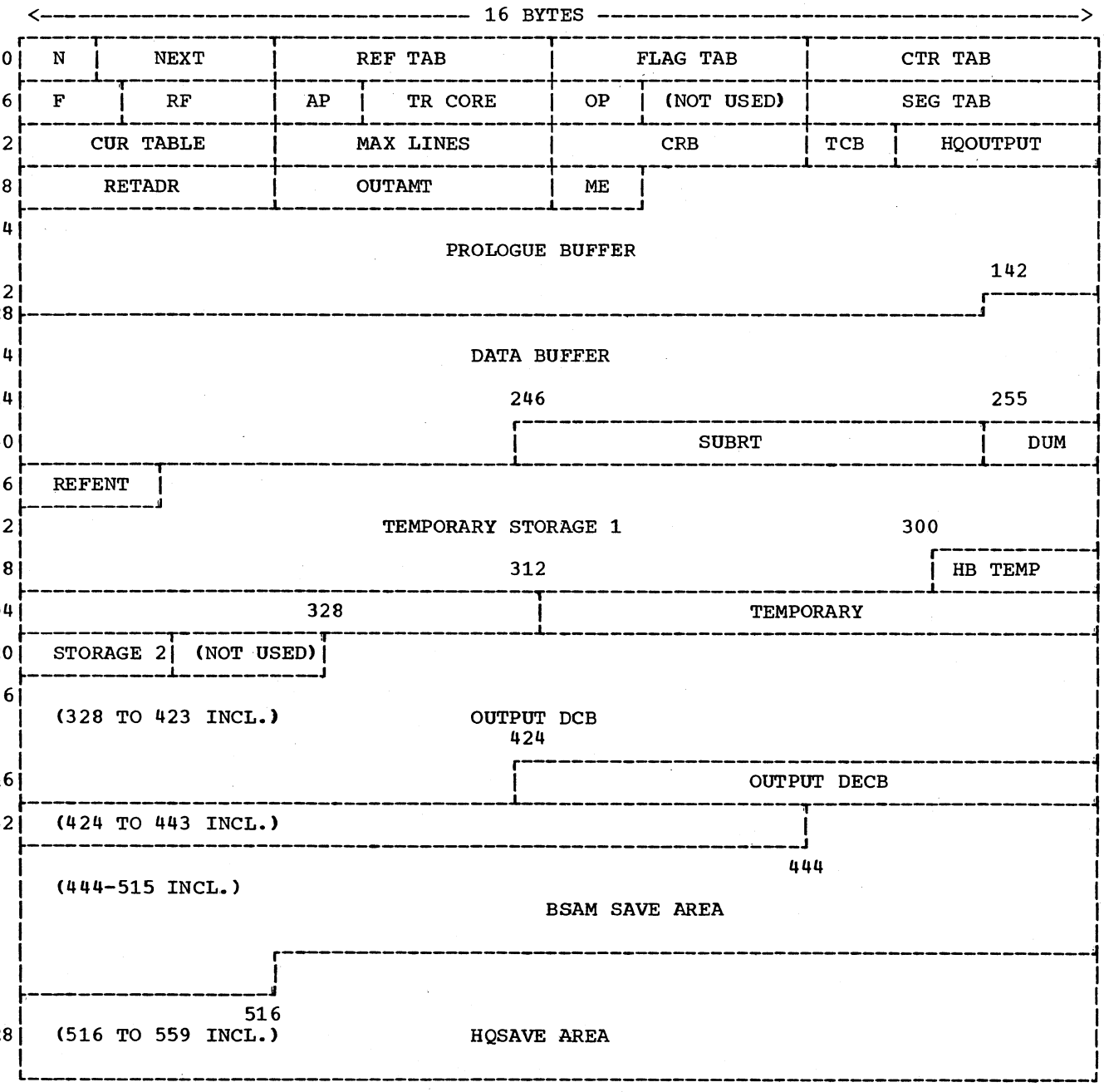
LN (1 byte) Length
Length of this entry in bytes

TA (3 bytes) TIA Table Address
This field contains the address of the TIA Table entry to be interpreted next.

APPENDIX B: TESTRAN INTERPRETER TABLES

This appendix contains diagrams and byte-by-byte descriptions of the tables generated and used by the TESTRAN interpreter.

CONTROL CORE 560 BYTES



N (1 Byte)
Last number assigned to a TIA table

NEXT (3 Bytes)
Address of next core area

REF TAB (4 Bytes)
Address of first reference table in the reference table chain

FLAG TAB (4 Bytes)
Address of first flag table in the flag table chain

CTR TAB (4 Bytes)
Address of first counter table in the counter table chain

F (2 Bytes) Flags
Bit 0 = Removed instruction execution control
0 = Execute removed instruction
1 = Delete execution of removed instruction
Bit 1 - Overlay flag
1 = Program contains overlay
0 = No overlay
Bits 2,3 = Table type
11 = Control
Bit 4 Link/load option
1 = Link mode
0 = Load mode
Bit 5 - Dump SEGTAB flag
1 = SEGTAB must be dumped
0 = No SEGTAB to be dumped
Bit 6 - Protection
1 = Protection
0 = No protection
Bit 7 - Floating point register flag
1 = Hardware contains floating-point
0 = No floating-point hardware
Bit 8 - No start trace flag
1 = Suppress trace start
0 = Permit trace start
Bit 9 - Not used
Bit 10 - SAVE routine flag
1 = SAVE routine has operated
0 = SAVE routine has not operated
Bit 11 - Trace mode
1 = Trace mode
0 = Non trace mode
Bit 12 - Branch type instruction (used by GO BACK)
1 = Instruction being executed is a branch
0 = Instruction being executed is not a branch
Bit 13 - Type Tasking
1 = Variable number of tasks
0 = Fixed number of tasks
Bit 14 - Not used
Bit 15 - Table opened
1 = A TIA has been opened
0 = No TIA opened

RF (2 Bytes) Return flags
0000 Return from all routines except TRACE, GO BACK, CLOSE
0008 Return from GO BACK, CLOSE
000E Return for TRACE, TESTRAN SVC
0014 Return from TRACE for TTOPEN or overlay SVC

AP (1 Byte) AT Priority
Priority from the last "TEST AT" macro

TR CORE (3 Bytes)
Address of working storage gotten for trace

OP (1 Byte) OPEN Priority
Priority from the "TEST OPEN" macro

NOT USED (3 Bytes)

SEG TAB (4 Bytes) Segment table address

CUR TABLE (4 Bytes) Current table address
Address of table core for the TIA table currently being interpreted

MAX LINES (4 Bytes)
Maximum lines of output to be generated for the current task

CRB (4 Bytes)
Callers R.B. address

TCB (1 byte)
Size of TCB in bytes

HQOUTPUT (3 Bytes)
Address of HQOUTPUT routine

RETADR (4 Bytes)
Address in users program control is to return to

OUTAMT (4 Bytes)
Number of lines of output already generated for this task

ME (2 Bytes)
Maximum execution count of TESTRAN macros to be interpreted during this task

PROLOGUE BUFFER (84 Bytes)

DATA BUFFER (104 Bytes)

SUBRT (9 Bytes)
Subroutine table

DUM (1 Byte)
Dummy GOBACK macro

REFENT (4 Bytes)
Address of the current reference table entry

TEMPORARY STORAGE 1 (40 Bytes)

HBTEMP (12 Bytes)
Temporary storage reserved for the HBADDRSR routine

TEMPORARY STORAGE 2 (12 Bytes)

NOT USED (4 Bytes)

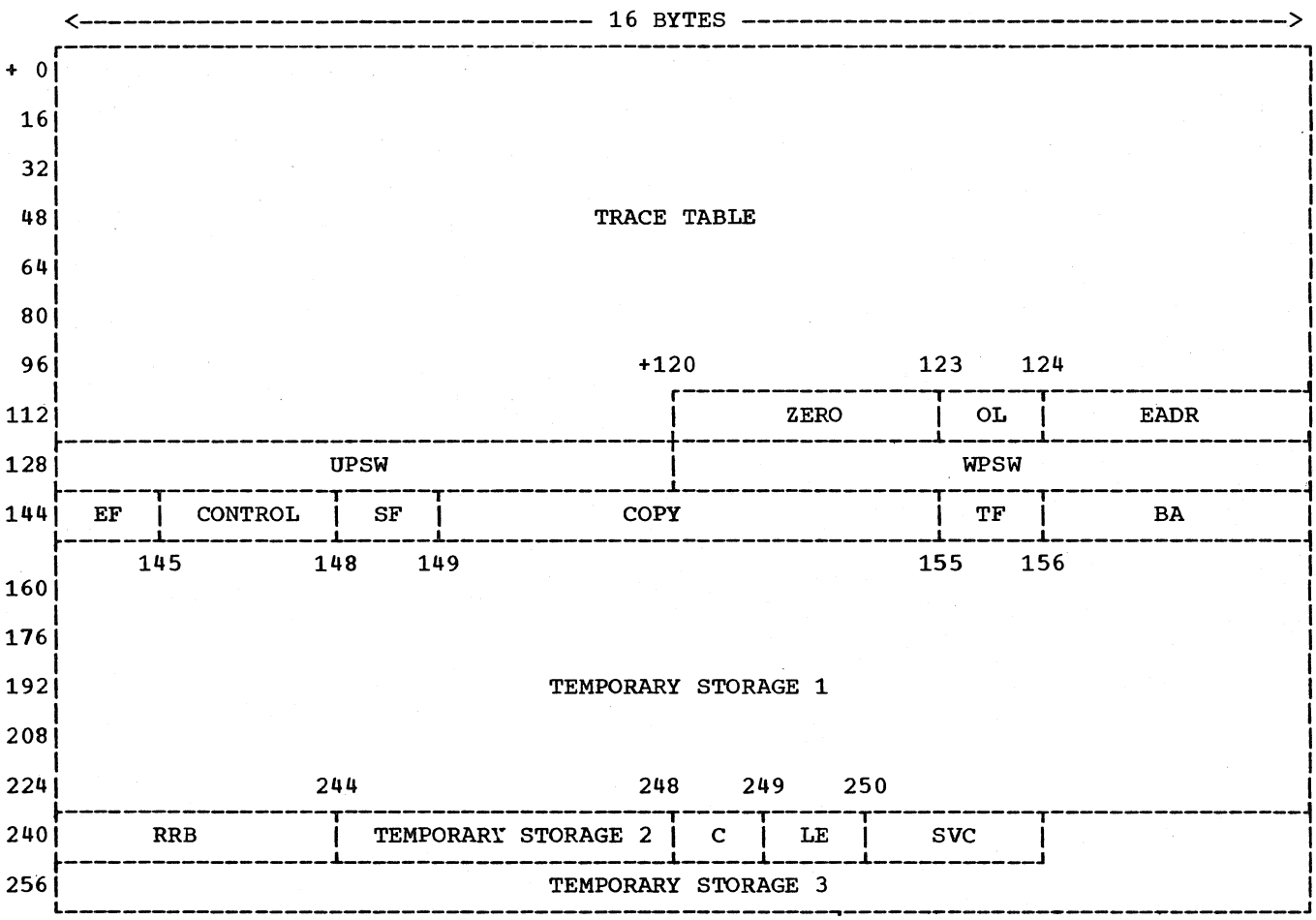
OUTPUT DCB (96 Bytes)
DCB to be used for intermediate output

OUTPUT DECB (20 Bytes)
DECB used for writing on the intermediate device

BSAM SAVE AREA (72 Bytes)
Register save area for the BSAM routines

HQSAVE AREA (44 Bytes)
Save area used by HQOUTPUT subroutine

TRACE CORE 272 BYTES



TRACE TABLE (120 BYTES)

This field contains a list of all currently active traces.

ZERO (3 Bytes)
Work constant of zero

OL (1 Byte) Output length
Storage for length of output data record

EADR (4 Bytes) Execute address
Address of last execute instruction traced

UPSW (8 Bytes) User PSW
Storage for the users PSW

WPSW (8 Bytes) Work PSW
PSW build area used during enable and disable functions

EF (1 Byte) Execute Flag
00 = Instruction being traced not an execute
FF = Instruction being traced is an execute

CONTROL (3 Byte) Control address

SF (1 Byte) Stop flags
Bits 0-3 Not used
Bit 4 1 = Stop trace due to SVC, bad EX, PRIV. instruction or bad store address
Bit 5 1 = Stop trace due to program check
Bit 6 1 = Stop trace due to overlay
Bit 7 1 = Stop trace due to TRACE STOP macro

COPY (6 Bytes) Instruction copy
Copy of the instruction presently being interpreted

TF (1 Byte) Flags
Bits 0,1 Not used
Bit 2 Interrupt routine flag
1 = User-supplied interrupt routine being traced
0 = Not in interrupt routine

Bit 3 SVC in switch
1 = TESTRAN SVC must be reinserted
0 = TESTRAN SVC not to be reinserted

Bit 4 Reference out switch
1 = The before portion of reference trace output has been generated for the instruction presently being interpreted
0 = No output generated

Bit 5 Reference trace switch
1 = Reference type trace active
0 = No reference type trace active

Bit 6 Call trace switch
1 = Call type trace active
0 = No call type trace active

Bit 7 Flow trace switch
1 = Flow type trace active
0 = No flow type trace active

BA (4 Bytes) Branch address
Effective address of branch instruction being interpreted

Temporary storage 1 (80 Bytes)

RRB (4 Bytes) Router RB
Contains the address of the TESTRAN routers routine block

Temporary storage 2 (4 Bytes)

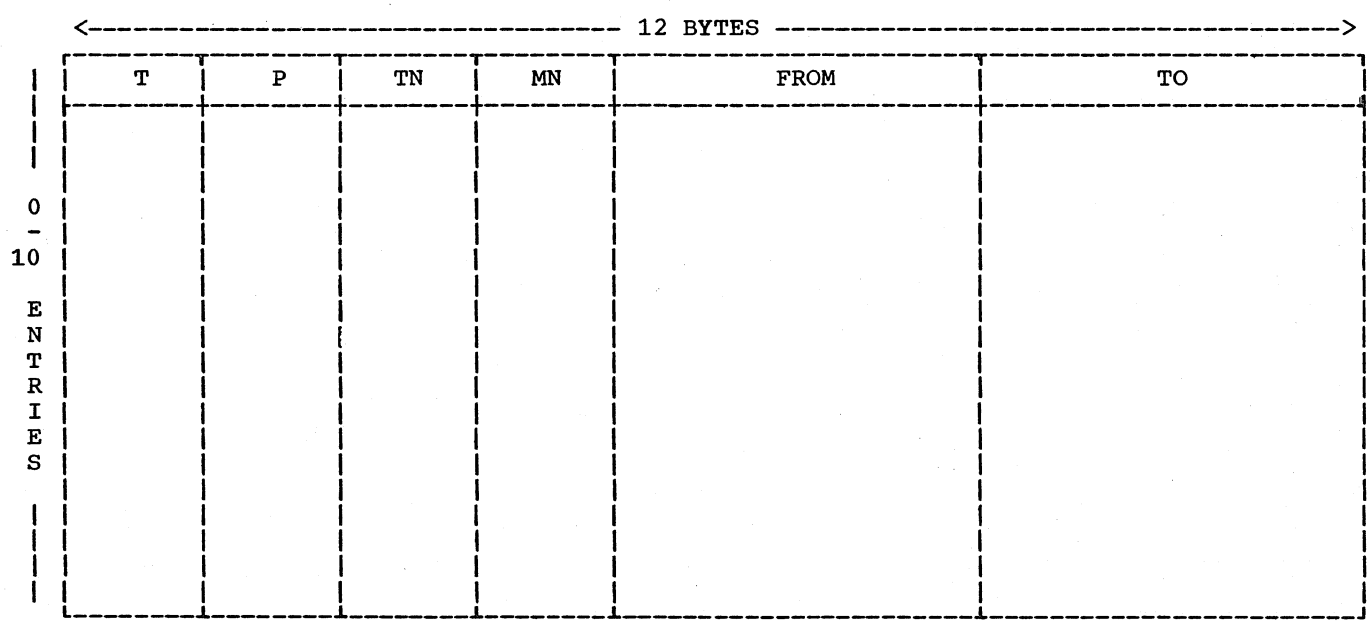
C (1 Byte) Count of trace table entries

LE (1 Byte) Last trace table entry

SVC (2 Bytes) Dummy SVC
Dummy SVC used during execution of a user SVC

Temporary storage 3 (20 Bytes)

TRACE TABLE (120 BYTES MAXIMUM)



T (1 Byte) Type of entry
00 = End of table
22 = Call type trace entry
26 = Reference type trace entry
2A = Flow type trace entry

P (1 Byte) Output Selection Code
Output selection code for output generated by this entry

TN (1 Byte) Table number
Number of the TIA table containing this trace macro

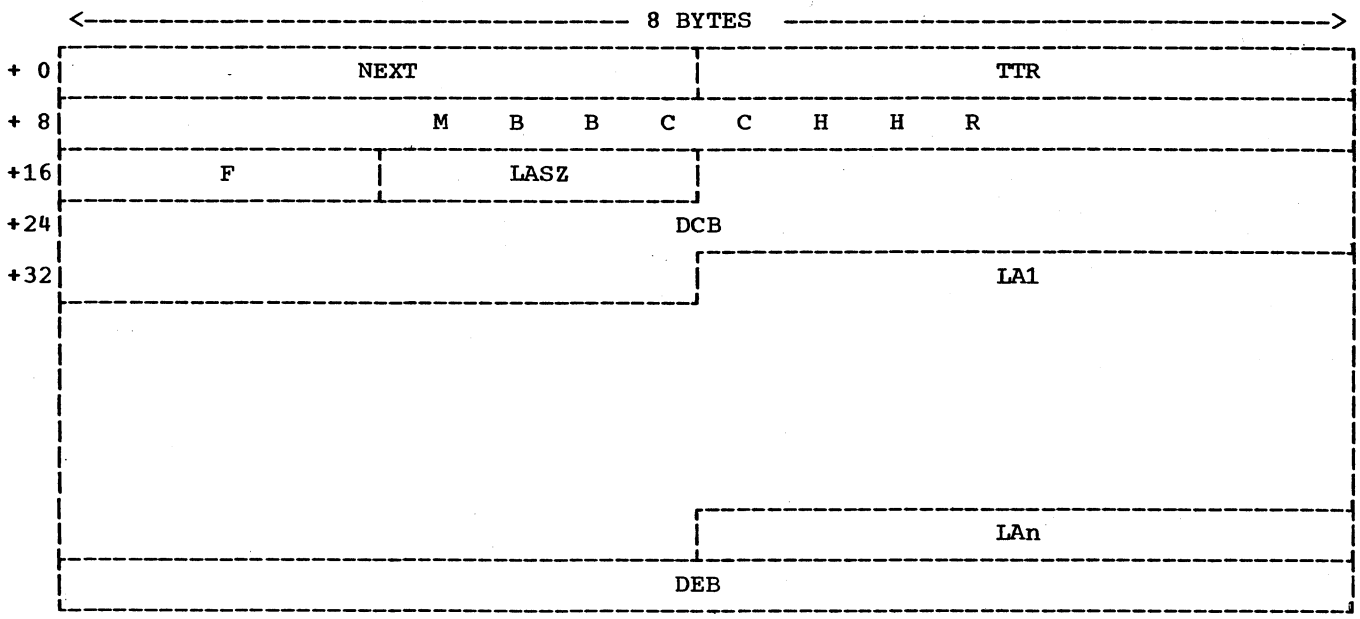
MN (1 Byte) Macro number
The macro number of the macro which generated this entry

FROM (4 Bytes)
Start address of the area being traced

TO (4 Bytes)

End address of the area being traced

DCB/REL CORE



NEXT (4 Bytes)
Pointer to the next DCB/REL. Zero in this field designates the end of the chain.

TTR (4 Bytes)
Relative disk address of the symbols, ESD and composite ESD

MBBCCHHR (8 Bytes)
Absolute disk address of the symbols, ESD and composite ESD

F (2 Bytes) Flags
Bits 0,1 Not used
Bits 2,3 Table type
10 = DCB/REL
Bits 4-15 Not used

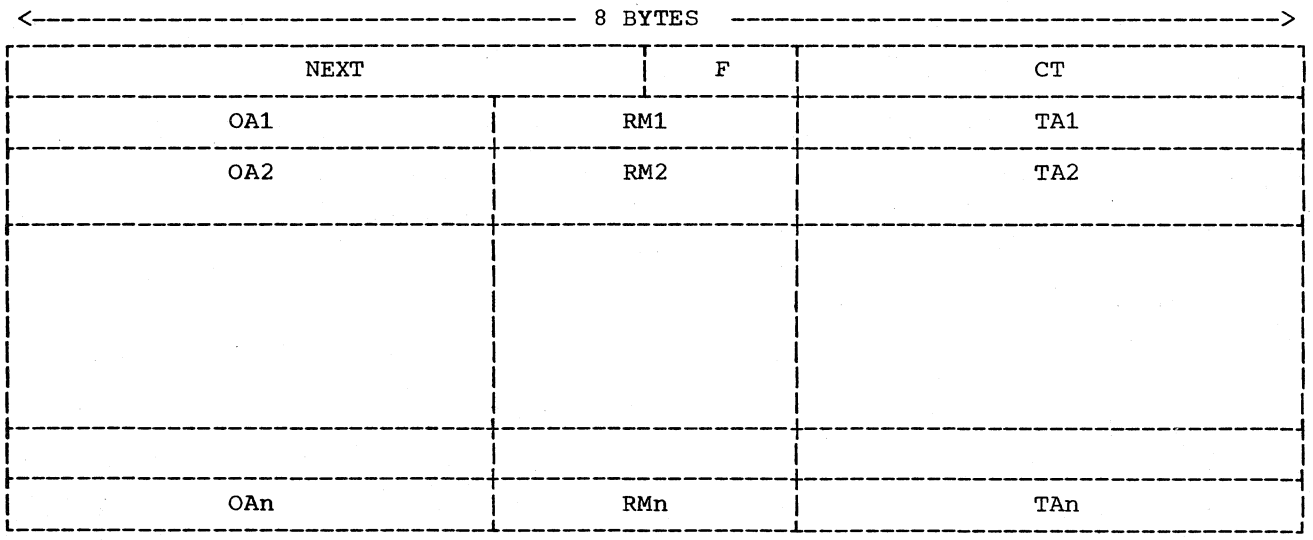
LASZ (2 Bytes)
This field contains a count of the bytes occupied by 'LA' entries.

DCB (16 Bytes)
This field contains a copy of the DCB used by program fetch to load the program.

LA1-LAn (4 Bytes each)
Loaded address of a control section in the load module

DEB (variable size)
Copy of the DEB used by program fetch in loading the program

REFERENCE TABLE



Each TESTRAN action table generates a reference table. Each SVC inserted, by TESTRAN, in the problem program generates an entry in the reference table.

NEXT (4 Bytes)
Address of the next reference table

F (1 Byte) Active/Inactive flag
00 = Table active
FF = Table inactive

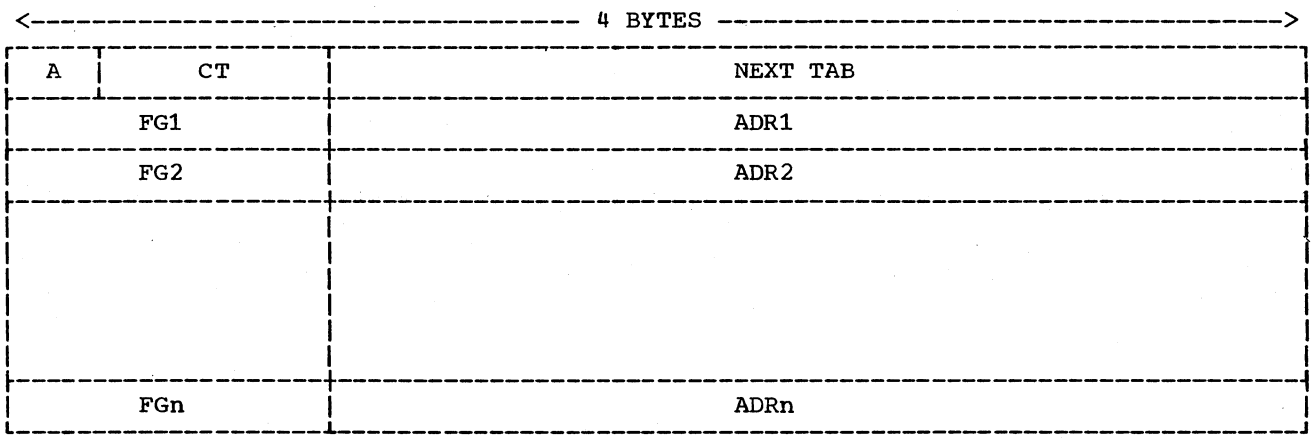
CT (3 Bytes)
Count of entries in this reference table

0A1-OAn (3 Bytes each) Object program address
Address in the object program where a TESTRAN SVC has been inserted

RM1-RMn (2 Bytes each) Removed instruction
The two bytes of the object program displaced by the TESTRAN SVC

TA1-TAn (3 Bytes each) TIA address
Address in the TESTRAN action table, of the 'TEST AT' macro which caused the SVC to be inserted

FLAG TABLE



A (1 Bit)
This bit specifies the table is active or inactive
0 = Active
1 = Inactive

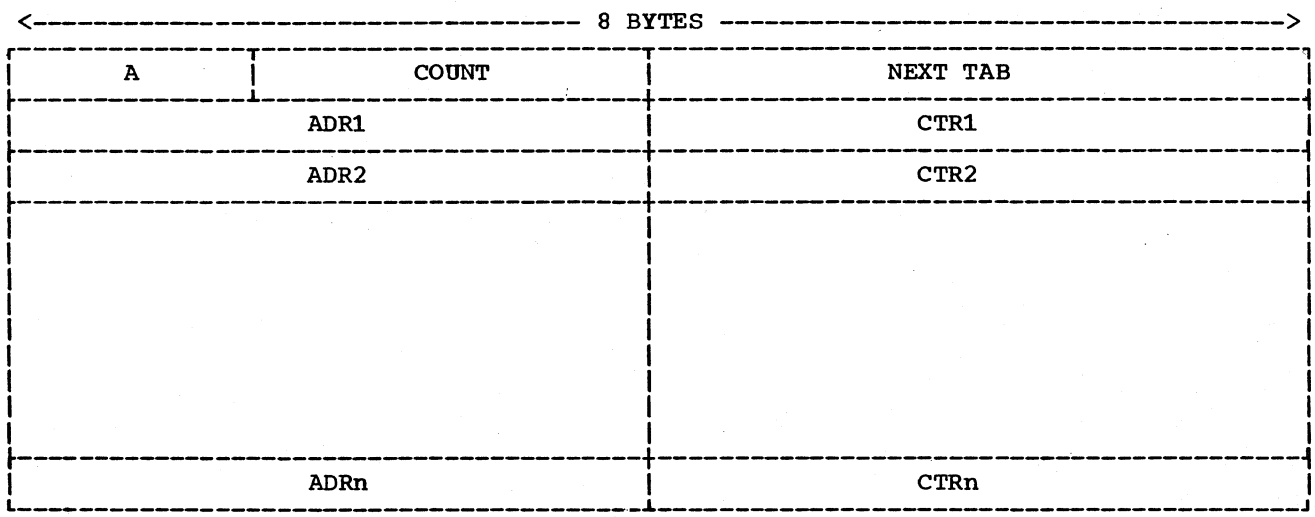
CT (7 Bits)
Count of the number of entries in this table

NEXT TAB (3 Bytes)
This field contains the address of the next flag table

FG1-FGn (1 Byte each)
The actual flag to be set and cleared by TESTRAN

ADR1-ADRn (3 Bytes each)
TIA table address of the flag

COUNTER TABLE



A (1 byte) Active byte
Zero-table is active
Not zero-table is inactive

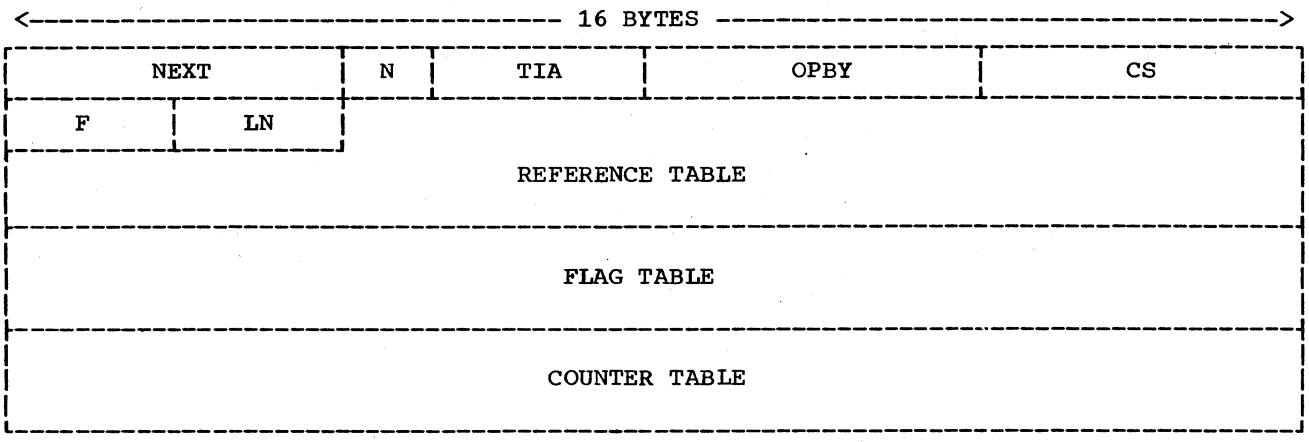
COUNT (3 Bytes)
Count of the number of entries in this table

NEXT TAB (4 Bytes)
This field contains the address of the next counter table in this chain.

ADR1 - ADRn (4 Bytes each)
These fields will contain the TESTRAN TIA table address of where the counter was defined.

CTR1 - CTRn (4 Bytes each)
These fields will contain the actual counters used by the TESTRAN routines.

TABLE CORE



NEXT (4 Bytes) Next pointer
Pointer to the next table core or DCB/REL core. Zero in this field indicates the end of the chain.

N (1 Byte) TIA number
Number assigned the TIA table associated with this table core

TIA (3 bytes) TIA address
Address of the TIA table for which this table core was generated

OPBY (4 Bytes) Open address
Address of the TEST OPEN macro which opened the TIA associated with this table core

CS (4 Bytes) Checksum
Checksum word for the TIA associated with this table core

F (2 Bytes) Flags
Bit 0 Active/Inactive Flag
0 = Table active
1 = Table inactive
Bit 1 Not used
Bits 2,3 Table type
00 = Table core
Bits 4-6 Not used
Bit 7 Just opened flag
0 = Just OPENed
1 = Not just OPENed
Bits 8-15 Not used

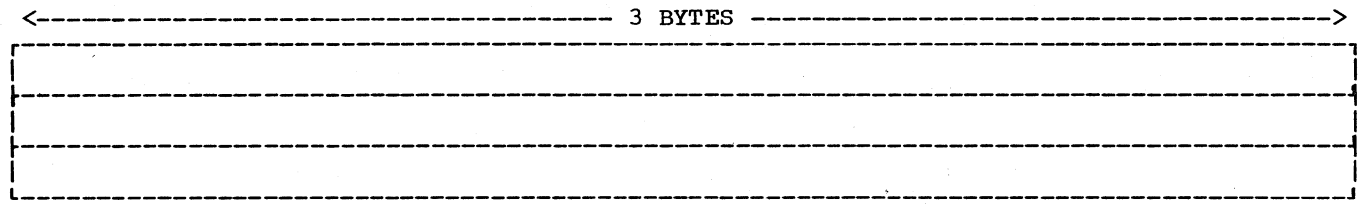
LN (2 Bytes) Length
Length of the TIA table for which this table core was generated

REFERENCE TABLE
Reference table associated with the TIA table

FLAG TABLE
Flag table associated with the TIA table

COUNTER TABLE
Counter table associated with the TIA table

SUBROUTINE TABLE



3 entries of 3 bytes each

The GO IN routine fills the first vacant 3 byte slot with a return address. If the table gets filled, (more than 3 entries) the first entry is pushed off the top and the whole table is moved up one entry with the new entry inserted at the bottom.

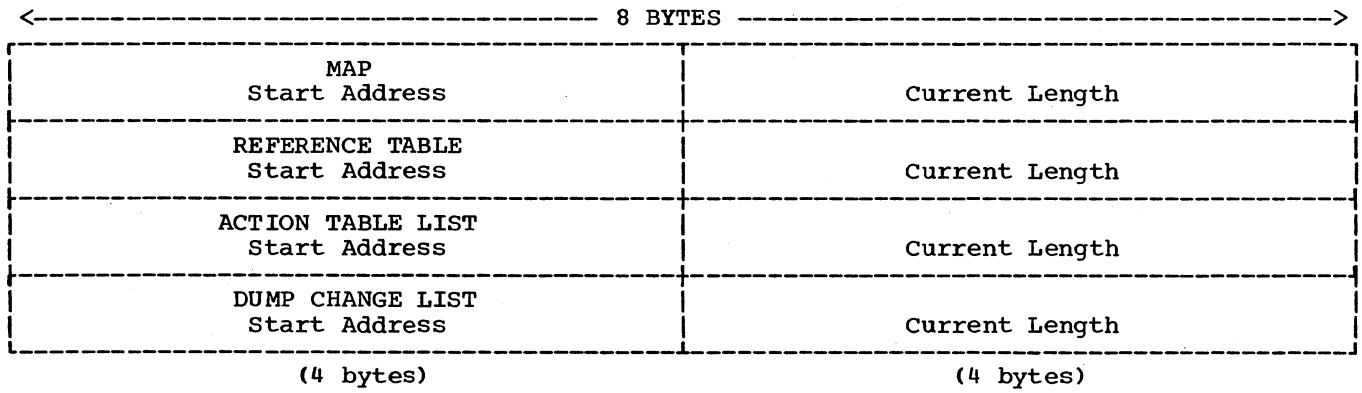
On returning, the addresses are used from bottom to top. As one is used for a return it is zeroed.

This entry table is zeroed on each entry to the TESTRAN router. (A TESTRAN SVC caused by a TEST AT macro.)

APPENDIX C: TESTRAN EDITOR TABLES

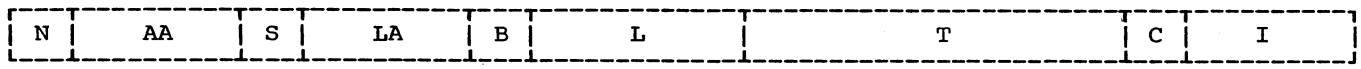
This appendix contains diagrams and byte-by-byte descriptions of the tables generated and used by the TESTRAN editor.

TABLE DICTIONARY



MAP ENTRY

(Sorted by LA field)



N (1 byte) Input Sequence Number

AA (3 bytes) Assembled Address

S (1 byte) Segment Number

LA (3 bytes) Loaded Address

B (1 byte) Status Byte
- Bit 0 Active Indicator
  - 0 = CSECT Active
  - 1 = CSECT Inactive
- Bits 4-7 Type
  - 0 = CSECT
  - 4 = Private Code
  - 5 = Common
  - 8 = DSECT

L (3 bytes) Length of CSECT in bytes

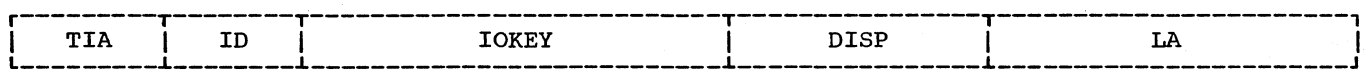
T (8 bytes) CSECT Name

C (1 byte) Assembly Number

I (3 bytes) I/O Key of Symbol Table

ACTION TABLE LIST

Entry



TIA (1 byte) Action Table Number

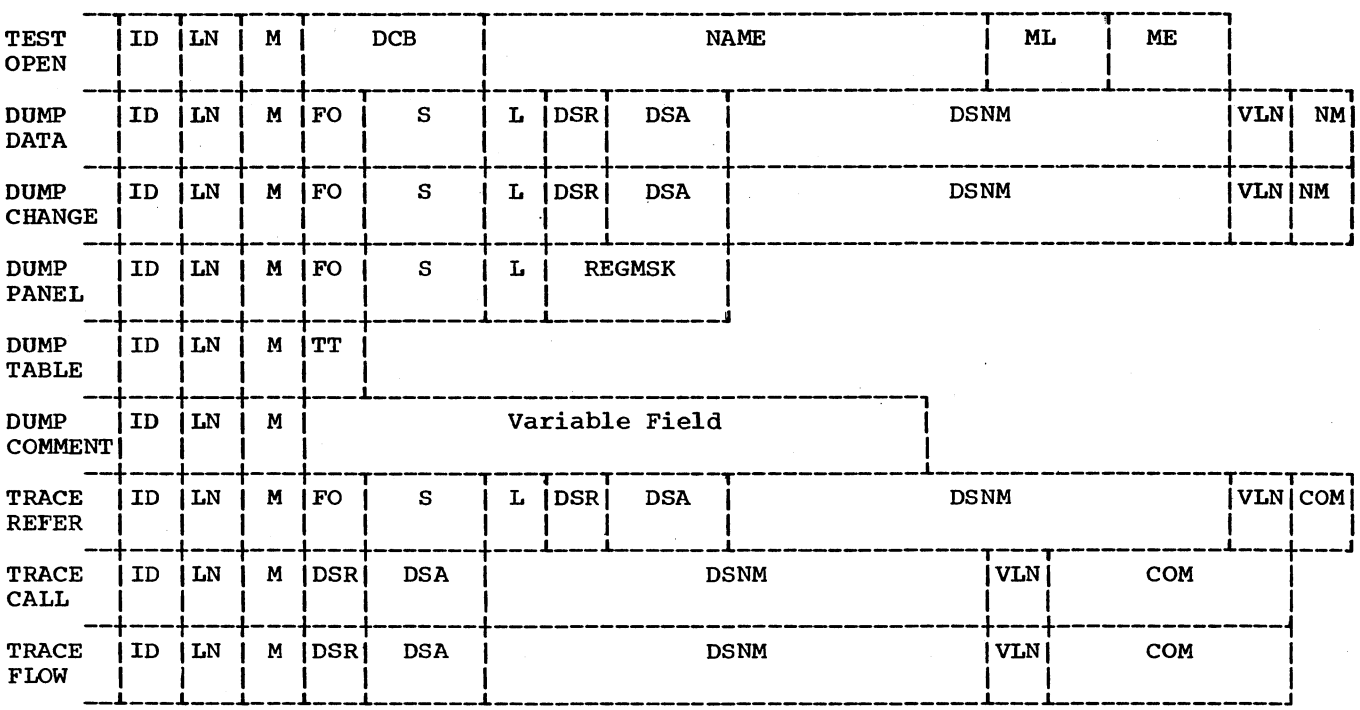
ID (1 byte) Identification of last entry in Action Table

IOKEY (3 bytes) I/O Key of Action Table

DISP (2 bytes) Offset of Action Table from beginning of buffer

LA (3 bytes) Loaded address of TIA Table at Interpreter Time

ACTION TABLE ENTRIES



ID (1 byte) Macro Identification Number

LN (1 byte) Length of entry in bytes

M (1 byte) Mask, indicating presence of
- Bit 0 FO or DCB
- Bit 1 S or ID
- Bit 2 L or ML
- Bit 3 DS or ME
- Bit 4 NM or COM
- Bit 5 REGMSK

DCB (3 bytes) Address of DCB

NAME (8 bytes) Name of TESTRAN Control Section

ML (2 bytes) Maximum number of output lines

ME (2 bytes) Maximum number of TESTRAN statements to execute

FO (1 byte) Format modifier
- 00 Character
- 01 Hexadecimal
- 02 Fixed Point
- 03 Floating Point
- 04 Packed Decimal
- 05 Zoned Decimal
- 0A Binary

S (2 bytes) Scale Modifier

L (2 byte) Length Modifier

DSR (1 byte) Repeat Count

DSA (2 bytes) Dummy Section Address

DSNM (8 bytes) Dummy Section Name

VLN (1 byte) Length of COM or N field

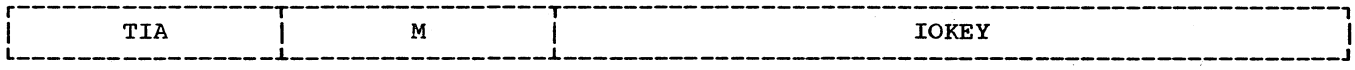
REGMSK (3 bytes) Register Mask
- Bits 0-15 General Registers 0-15
- Bit 16 Floating Point Register 0
- Bit 18 Floating Point Register 2
- Bit 20 Floating Point Register 4
- Bit 22 Floating Point Register 6

TT (1 byte) Table ID Dumped

COM (Variable) Comment Over-rider

NM (Variable) Name Over-rider

DUMP CHANGE LIST ENTRY



TIA (1 byte) Action Table Number

M (1 byte) Macro Identification Number

IOKEY (3 bytes) I/O Key of Dump Change Table

DUMP CHANGE TABLE ENTRY

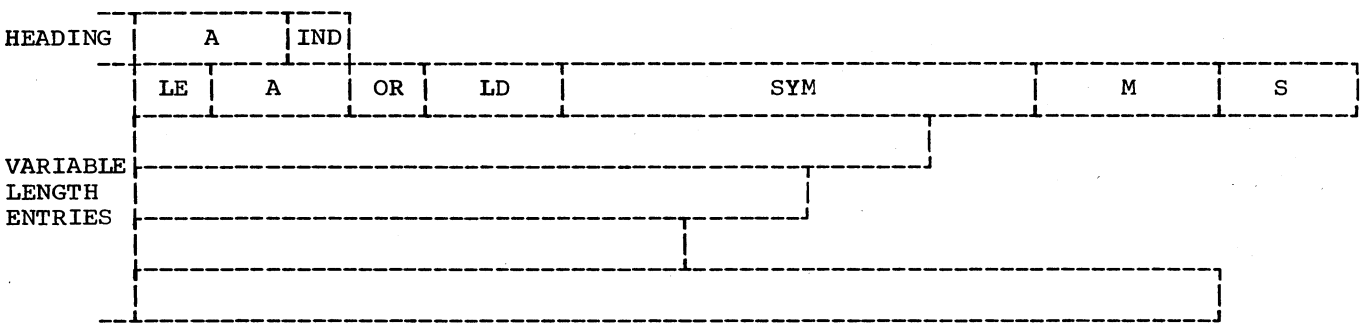
N (1 byte) Dump Change Table Entry Number

LA (3 bytes) Loaded Address of Data

L (2 bytes) Length of Data

IOKEY (3 bytes) I/O Key of Dumped Data

SYMBOL TABLE



HEADING (5 bytes) One heading per buffer

A (2 bytes) Offset from the start of the Section Definition for the last byte defined in this buffer

IND (1 byte) Indicator for last buffer
FF = More SYMS for this Section Definition
00 = No more SYMS for this Section Definition

VARIABLE LENGTH ENTRY (variable) One entry per named or data defining statement

LE (1 byte) Length of this entry

A (2 bytes) Offset from the start of the Section Definition of the first byte described by this entry

OR (1 byte) Organization
Bits 0-3 Format
0 Character
1 Hex
2 Fixed Point
3 Floating Point
4 Packed Decimal
5 Zoned Decimal
6 A Type Data
7 Y Type Data
8 S Type Data
9 V Type Data
A B Type Data
B Instruction
C CCW
D Space

Bit 4 Zero if label present
One if label not present

Bits 5-6 Duplication
00 No Duplication, not cluster subfield
01 Cluster subfield
10 Duplication, no cluster
11 Duplication, with cluster subfields

Bit 7 Zero if no scaling
One if scaling

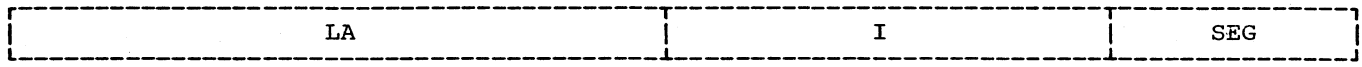
LD (2 bytes) Explicit length of data field

SYM (0 or 8 bytes) Label

M (0 or 3 bytes) Duplication factor

S (0 or 2 bytes) Scaling

REFERENCE TABLE ENTRY



LA (3 bytes) Loaded address of displaced instruction

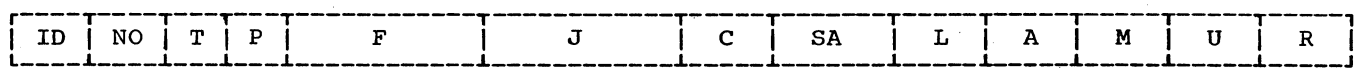
I (2 bytes) Displaced instruction

SEG (1 byte) Overlay segment number containing TESTRAN SVC

APPENDIX D: TESTRAN EDITOR INPUT RECORD FORMATS

This appendix contains diagrams of the organization of the 21 record types generated by the TESTRAN interpreter and processed by the TESTRAN editor. The presentation for the prologue record of the record pairs is generalized, while the data records are presented separately for each record type.

PROLOGUE RECORD



ID (1 byte) Record Identification
00000000 Skip this record
00000010 TESTRAN

NO (1 byte) For Type 5E records - TIA table identification number
For Type 5A records - Segment number or zero
For Types 22, 26, 2A records
Bit 0 = 0 - Normal trace record
1 - Trace start record
For Type 1E records
04 = DCB
08 = DEB
0C = TCB, where
Bit 0 = 0 - No floating point registers
1 - floating point included

T (1 byte) Type of Entry in D field (See note below)
06 DUMP DATA
0A TEST OPEN
0E DUMP CHANGES
12 DUMP MAP
16 DUMP PANEL
1A DUMP COMMENT
1E DUMP TABLE
22 TRACE CALL
26 TRACE REFER
2A TRACE FLOW
2E TRACE STOP
32 TEST CLOSE
36 ERROR
42 Continuation
46 Message
4A CESD
4E Map Change
52 CSECT Relocation Table
56 Symbol Table
5A Reference Table
5E TIA Table

Note: Bit 7 in the T byte, if one, indicates that the current Data Record is continued in the next Data Record.

P (1 byte) TESTRAN Output Selection Code (8 bits, left to right, indicate output selections 1-8 respectively)

F (4 bytes) where
Byte 0 - Action Table ID number of Action Statement
Bytes 1-3 - Address of AT statement (null if no AT)

J (8 bytes) Name of load module

C (1 byte) Number of M fields

SA (3 bytes)
Absolute starting address of data for 06, 0E, 1E or TIA Table in Type 5A, 5E records
Ignored for all other entry types

L (2 bytes)
Length of Data Field (bytes) including continuations
Exception: Trace with continuation. In this case, the length is for the current Data Record only.

A (2 bytes) Current Action Pointer
Byte 0 - Action Table ID Number
Byte 1 - Macro Number

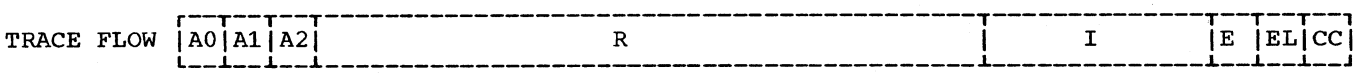
M (2 bytes) Executed Statement Pointer
Byte 0 - Action Table ID number
Byte 1 - Macro Number
The M fields provide the ability to supply the programmer with a trace of the executed TESTRAN statements prior to the action which resulted in this output record.

Note: The data record for asynchronous trace output starts in the third byte of the M field and is continued in the Data Record. There are no TESTRAN executed statements associated with asynchronous trace output. This minimizes continuation in Trace Refer.

U (14 bytes) Not used
Regardless of the number of M fields, this field always starts with byte 85 (offset from beginning of record by 84 bytes).

R (4 bytes)
Contents of the base register for DSECT, used in DUMP DATA and DUMP CHANGES.

M FIELD EXPANSION



A0 (2 bytes) Not used

A1 (4 bytes) From Address (absolute)

A2 (4 bytes) To Address (absolute)

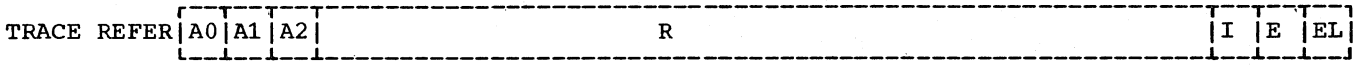
R (64 bytes) General Registers 1 thru 0

I (8 bytes) Active Branch Instruction

E (4 bytes) Execute Instruction
(null if not present)

EL (4 bytes) Location of execute instruction (absolute)
(not present if E is null)

CC (4 bytes) Program Status Word



A0 (2 bytes) Not used

A1 (4 bytes) Referencing Location (absolute)

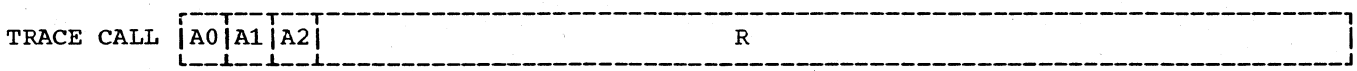
A2 (4 bytes) Referencing Location (absolute)

R (64 bytes) General Registers 1 thru 0

I (8 bytes) Referencing Instruction

E (4 bytes) Execution instruction
(null if not executed)

EL (4 bytes) Location of execute instruction (absolute)
(not present if E is null)



A0 (2 bytes) Not used

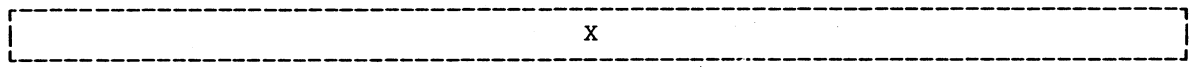
A1 (4 bytes) From Address (absolute)

A2 (4 bytes) To Address (absolute)

R (64 bytes) General Registers 1 thru 0

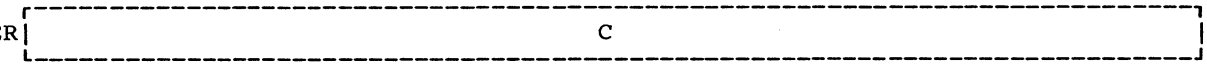
DATA RECORD

TRACE FLOW



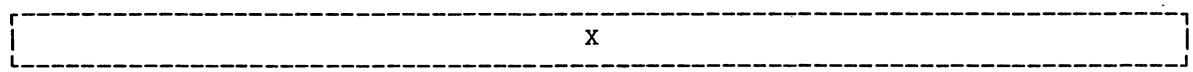
X (16 bytes) Padded Dummy Record
(All data in M field of Prologue)

TRACE REFER



C (variable) Referenced Data
Contents of referenced location before reference (256 bytes maximum). The C field in the next sequential Data Record will contain the contents of the referenced location after reference.

TRACE CALL



X (16 bytes) Padded dummy record
(All data in M field of Prologue)

DUMP MAP

C (1 byte)
Number of T, N, A1, A2, fields in the Data Record

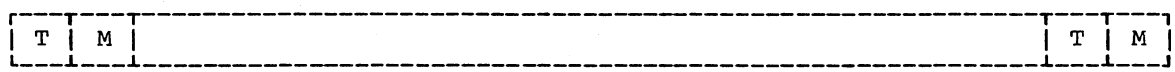
T (1 byte) Type Area
01 = Program area
02 = Data area

N (8 bytes) Name of Task

A1 (4 bytes) Beginning actual address of area to be mapped

A2 (4 bytes) Ending actual address of area to be mapped

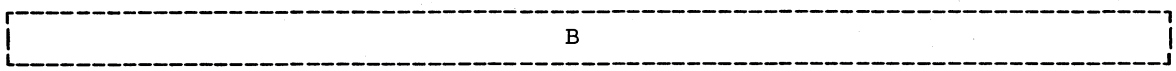
TRACE STOP



T (1 byte) Action ID of action stopped

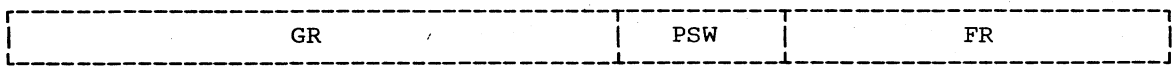
M (1 byte) Macro ID of action stopped

DUMP DATA
DUMP CHANGES



B (variable) Dumped Data
The starting address of the Dump is found in the SA field of the prologue record.

DUMP PANEL

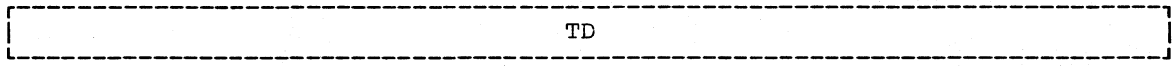


GR (64 bytes) General Registers 0-15

PSW (8 bytes) Program status word

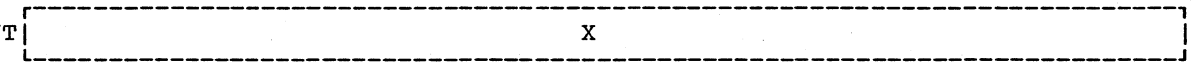
FR (32 bytes) Floating point registers 0-7
(Will not be present if file was generated on system with no floating point option.)

DUMP TABLE



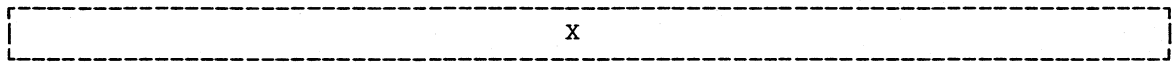
TD (variable) Contents of table.
The table ID is found in the NO field in the prologue record.

DUMP COMMENT



X (16 bytes) Padded Dummy Record

TEST OPEN

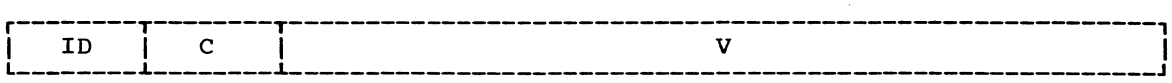


X (16 bytes) Padded dummy record

TEST CLOSE

N (1 byte) Number of Action Table closed

ERROR Message

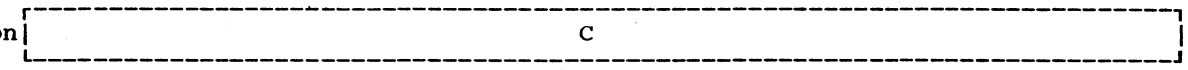


ID (2 bytes) Message ID Number

C (2 bytes) Count of characters in V field. If 128 or greater, the V field must be edited. If bit 9=1, the V field is converted to Hex; otherwise, it is converted to decimal.

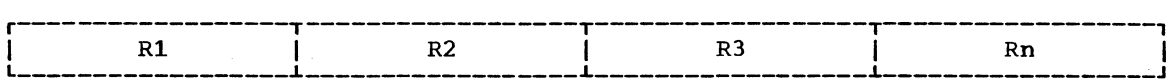
V (variable) Characters to add to message

Continuation



C (variable) Continuation of previous Data Record

Relocation Table

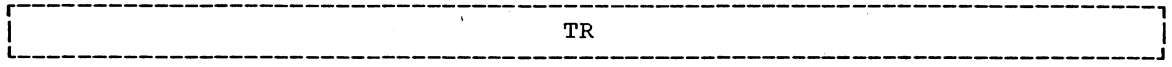


R (4 bytes) Relocated Address

Block Load
One 4 byte relocated address (indicates this relocation factor is applied to all control sections of this linked module).

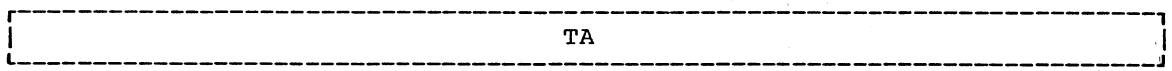
Scatter Load
A 4 byte relocated address for each respective control section of this linked module.

Reference Table



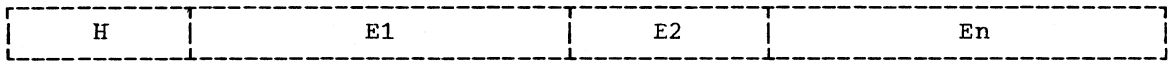
TR (variable) Reference Table
(See Appendix B for detail.)

TIA Table



TA (variable) TIA Table
(See Appendix A for detail.)

CESD

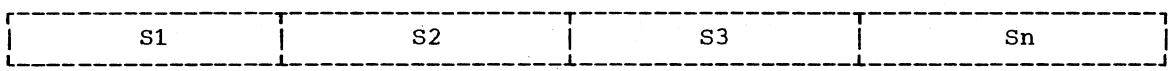


H (8 bytes) Header - Ignored

E (16 bytes) CESD entry

Bytes 0-7 Control Section Name
Byte 8 ESD Type
Bytes 9-11 Linkage Edited Address
Byte 12 Segment Number
Bytes 13-15 Control Section Length

Map CHANGE

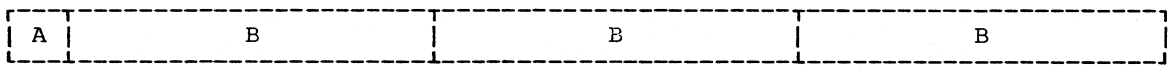


S (4 bytes) Overlay Segment Indicator

Bit 0-30 Ignored
Bit 31 If zero, segment is in storage and is active.

Note: Each 4 byte entry occurs in numerical sequence.

Symbol Table



A (4 bytes) Header

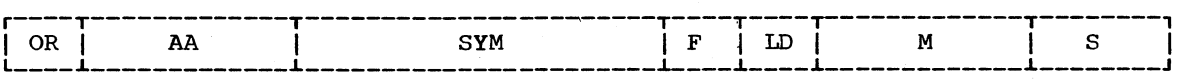
Bytes 0-2 Not Used
Byte 3 Byte Count on remainder of record (240 bytes max.)
Bytes 4-9 Not Used
Bytes 10-11 Byte count of text in this card image
Bytes 16-71 Symbol Table text
Bytes 72-79 Not Used

B (80 bytes) Card Images (3 maximum)

Byte 0 Hex 02
Bytes 1-3 Characters SYM

Note: Bytes 16-71 of the B field (Symbol Table Text) are further described as follows:

Symbol Table Text



OR (1 byte) Organization Byte

Bit 0 - If 0, not Data Type. Bits 1-3 indicate the following:

Bits 1-3
- 000 Space
- 001 Control Section
- 010 Dummy Control Section
- 011 Common
- 100 Instruction
- 101 Command Control Word
- 110 Not Used
- 111 Not Used

- If 1, Data Type. Bits 1-3 indicate the following:

Bit 1
If 0, no duplication
If 1, duplication (indicates presence of M field)

Bit 2
If bit 1=0
0 indicates independent
1 indicates cluster subfield
If bit 1=1
0 indicates independent
1 indicates cluster

Bit 3
If 0, no scaling
If 1, scaling (indicates presence of S field)

Bit 4 - If 0, label present
If 1, label not present

Bits 5-7 Length of label minus one

AA (3 bytes) Displacement from base of Control Section

SYM (0-8 bytes) Symbol

F (1 byte) Format
- 00 Character
- 04 Hex
- 08 B Type Data
- 0C Not Used
- 10 Fixed Point, Full
- 14 Fixed Point, Half
- 18 Floating Point, Short
- 1C Floating Point, Long
- 20 A Type Data
- 24 Y Type Data
- 28 S Type Data
- 2C V Type Data
- 30 Packed Decimal
- 34 Zoned Decimal

LD (1 byte) Data Length (actual length minus 1)
If the format of the data is character, binary, or hex, the LD field will be two bytes.

M (3 bytes) Duplication factor

S (2 bytes) Scaling

APPENDIX E: TESTRAN FLOWCHART CROSS REFERENCE LIST

This appendix lists all TESTRAN routines by their symbolic names and all TESTRAN macro instructions by name. It gives the flowchart reference for each item and names the function of each item.

A routine whose name is preceded by an asterisk contains message text.

ROUTINE	FLOWCHART	PROGRAM	FUNCTION
DUMP	13-A1		MACRO-DEFINITION
GO	12-A1		MACRO-DEFINITION
HBADDRSR	34-A2	INTERPRETER	ROUTER SUBROUTINE
HQOUTPUT	34-A5	INTERPRETER	ROUTER SUBROUTINE
IGC0004I	31-A2	INTERPRETER	TTOPEN1 SVC 49
IGC0106A	30-A2	INTERPRETER	SAVE
IGC038	30-A5	INTERPRETER	RESIDENT SVC
* IEGMC00A	50-A2	EDITOR	START
IEGME00A	50-E4	EDITOR	EDITOR ROUTER
* IEGMG00A	54-A2	EDITOR	ACTION ROUTER
IEGNA00A	61-A3	EDITOR	DUMP DATA
* IEGND00A	62-A2	EDITOR	DUMP CHANGES
IEGNG00A	59-A2	EDITOR	DUMP MAP
IEGNM00A	59-H1	EDITOR	DUMP COMMENT
IEGNP00A	59-F4	EDITOR	DUMP TABLE
IEGNS00A	55-C1	EDITOR	SYMBOL TABLE INITIALIZER
IEGNV00A	60-A1	EDITOR	TRACE
IEGNY00A	55-A4	EDITOR	SYMBOL TABLE, FIRST PASS
IEGOPEN2	32-A2	INTERPRETER	TTOPEN2 COPY
IEGOPEN3	32-A4	INTERPRETER	TTOPEN3 OPEN
IEGPA00A	59-A5	EDITOR	TRACE STOP
* IEGPE00A	53-A1	EDITOR	INVALID RECORD
IEGPEDIT	51-A2	EDITOR	PANEL EDIT
IEGPG00A	61-A5	EDITOR	TEST OPEN
IEGPH00A	53-A3	EDITOR	TEST CLOSE
* IEGPI00A	53-E4	EDITOR	INTERPRETER MESSAGE
* IEGPK00A	53-F2	EDITOR	END OF RUN
IEGPP00A	61-A1	EDITOR	DUMP PANEL
IEGRA00A	53-A5	EDITOR	CESD MAP
* IEGRC00A	54-A4	EDITOR	MAP CHANGE
* IEGRE00A	52-A3	EDITOR	RELOCATION TABLE
* IEGRF00A	55-A2	EDITOR	SYMBOL TABLE BASE
IEGRG00A	58-A5	EDITOR	SYMBOL TABLE, ESD@S
* IEGRK00A	52-E5	EDITOR	REFERENCE TABLE
* IEGRL00A	52-F2	EDITOR	ACTION TABLE
IEGSF00A	51-A4	EDITOR	EDITOR MESSAGE
IEGSN00A	64-A1	EDITOR	ADDRESS ANALYZER
IEGSP00A	57-A3	EDITOR	SYMBOL TABLE, LAST PASS
IEGSQ00A	64-A4	EDITOR	SYMBOL SEARCH
IEGSR00A	65-A2	EDITOR	ATTRIBUTE ANALYZER
IEGSU01Z	66-C2	EDITOR	EDIT, HEX
IEGSU06Z	66-G1	EDITOR	EDIT, INSTRUCTION
IEGSU40Z	66-C5	EDITOR	EDIT, ALPHAMERIC
IEGSU50Z	67-A1	EDITOR	EDIT, BINARY
IEGSU60Z	66-A4	EDITOR	EDIT, ZONED DECIMAL
IEGSU70Z	66-A1	EDITOR	EDIT, PACKED DECIMAL
IEGSU80Z	51-E3	EDITOR	EDIT, FIXED POINT
IEGSU90Z	67-A3	EDITOR	EDIT, FLOATING POINT

ROUTINE	FLOWCHART	PROGRAM	FUNCTION
IEGTTRNA	40-A1	INTERPRETER	DUMP DATA
IEGTTRNB	40-H2	INTERPRETER	DUMP COMMENT
IEGTTRNC	40-H1	INTERPRETER	DUMP PANEL
IEGTTRND	40-A4	INTERPRETER	GO IN/OUT/TO
IEGTTRNE	36-A4	INTERPRETER	TEST ON
IEGTTRNF	31-A4	INTERPRETER	DUMP TABLE
IEGTTRNG	37-A1	INTERPRETER	TEST WHEN
IEGTTRNH	39-A1	INTERPRETER	TEST CLOSE
IEGTTRNJ	41-A2	INTERPRETER	GO BACK
IEGTTRNK	40-F4	INTERPRETER	DUMP MAP
IEGTTRNL	37-A3	INTERPRETER	TRACE START
IEGTTRNM	37-A4	INTERPRETER	TRACE STOP
IEGTTRNN	42-A4	INTERPRETER	SET COUNTER
IEGTTRNO	35-A2	INTERPRETER	OVERLAY 1
IEGTTRNP	42-A2	INTERPRETER	SET FLAG
IEGTTRNR	39-A4	INTERPRETER	SET VARIABLE
IEGTTRNT	43-A1	INTERPRETER	TRACER
IEGTTRNX	36-A2	INTERPRETER	OVERLAY 2
IEGTTRNZ	38-A2	INTERPRETER	TRACE INTERRUPT
IEGTTROT	33-A2	INTERPRETER	ROUTER
JAMACHCK	43-H1	INTERPRETER	MACHINE CHECK INTERRUPT
JAPROGCK	43-H2	INTERPRETER	PROGRAM CHECK INTERRUPT
SET	11-A1		MACRO-DEFINITION
TEST	10-A2		MACRO-DEFINITION
TRACE	15-A1		MACRO-DEFINITION

APPENDIX F: TESTRAN CONTROL BLOCK AND RECORD FORMAT CROSS REFERENCE LIST

This appendix lists all of TESTRAN's control blocks and record formats. Each item listed is identified with the portion of TESTRAN which uses or processes it. It is also identified with the appendix and page on which it is presented.

CONTROL BLOCK, RECORD	PROGRAM	APPENDIX
ACTION TABLE	EDITOR	C - Pg. 110
ACTION TABLE LIST	EDITOR	C - Pg. 109
CONTROL CORE	INTERPRETER	B - Pg. 96
COUNTER TABLE	INTERPRETER	B - Pg. 105
DATA INPUT, CESD	EDITOR	D - Pg. 119
DATA INPUT, CONTINUATION	EDITOR	D - Pg. 118
DATA INPUT, DUMP CHANGES	EDITOR	D - Pg. 117
DATA INPUT, DUMP COMMENT	EDITOR	D - Pg. 117
DATA INPUT, DUMP DATA	EDITOR	D - Pg. 117
DATA INPUT, DUMP MAP	EDITOR	D - Pg. 116
DATA INPUT, DUMP PANEL	EDITOR	D - Pg. 117
DATA INPUT, DUMP TABLE	EDITOR	D - Pg. 117
DATA INPUT, ERROR MESSAGE	EDITOR	D - Pg. 118
DATA INPUT, MAP CHANGE	EDITOR	D - Pg. 119
DATA INPUT, REFERENCE TABLE	EDITOR	D - Pg. 118
DATA INPUT, RELOCATION TABLE	EDITOR	D - Pg. 118
DATA INPUT, SYMBOL TABLE	EDITOR	D - Pg. 119
DATA INPUT, TEST CLOSE	EDITOR	D - Pg. 117
DATA INPUT, TEST OPEN	EDITOR	D - Pg. 117
DATA INPUT, TIA TABLE	EDITOR	D - Pg. 118
DATA INPUT, TRACE CALL	EDITOR	D - Pg. 116
DATA INPUT, TRACE REFER	EDITOR	D - Pg. 116
DATA INPUT, TRACE STOP	EDITOR	D - Pg. 116
DATA INPUT, TRACE FLOW	EDITOR	D - Pg. 116
DCB/REL CORE	INTERPRETER	B - Pg. 102
DUMP CHANGE LIST	EDITOR	C - Pg. 111
DUMP CHANGE TABLE	EDITOR	C - Pg. 111
FLAG TABLE	INTERPRETER	B - Pg. 104
MAP	EDITOR	C - Pg. 108
PROLOGUE INPUT, TRACE FLOW	EDITOR	D - Pg. 115
PROLOGUE INPUT, TRACE REFER	EDITOR	D - Pg. 115
PROLOGUE INPUT, TRACE CALL	EDITOR	D - Pg. 115
REFERENCE TABLE	EDITOR	C - Pg. 113
REFERENCE TABLE	INTERPRETER	B - Pg. 103
SYMBOL TABLE	EDITOR	C - Pg. 112
SUBROUTINE TABLE	INTERPRETER	B - Pg. 107
TABLE CORE	INTERPRETER	B - Pg. 106
TABLE DICTIONARY	EDITOR	C - Pg. 108
TIA TABLE, A FIELD	INTERPRETER	A - Pg. 81
TIA TABLE, DUMP CHANGES	INTERPRETER	A - Pg. 84
TIA TABLE, DUMP COMMENT	INTERPRETER	A - Pg. 86
TIA TABLE, DUMP DATA	INTERPRETER	A - Pg. 83
TIA TABLE, DUMP MAP	INTERPRETER	A - Pg. 84
TIA TABLE, DUMP PANEL	INTERPRETER	A - Pg. 85
TIA TABLE, DUMP TABLE	INTERPRETER	A - Pg. 86
TIA TABLE, GO BACK	INTERPRETER	A - Pg. 95
TIA TABLE, GO IN	INTERPRETER	A - Pg. 94
TIA TABLE, GO OUT	INTERPRETER	A - Pg. 95
TIA TABLE, GO TO	INTERPRETER	A - Pg. 95

CONTROL BLOCK, RECORD	PROGRAM	APPENDIX
TIA TABLE, TEST AT	INTERPRETER	A - Pg. 89
TIA TABLE, TEST CLOSE	INTERPRETER	A - Pg. 90
TIA TABLE, TEST DEFINE COUNTER	INTERPRETER	A - Pg. 91
TIA TABLE, TEST DEFINE FLAG	INTERPRETER	A - Pg. 91
TIA TABLE, TEST ON	INTERPRETER	A - Pg. 92
TIA TABLE, TEST OPEN	INTERPRETER	A - Pg. 90
TIA TABLE, TEST WHEN	INTERPRETER	A - Pg. 93
TIA TABLE, TRACE CALL	INTERPRETER	A - Pg. 88
TIA TABLE, TRACE FLOW	INTERPRETER	A - Pg. 88
TIA TABLE, TRACE REFER	INTERPRETER	A - Pg. 87
TIA TABLE, TRACE STOP	INTERPRETER	A - Pg. 89
TIA TABLE, SET COUNTER	INTERPRETER	A - Pg. 93
TIA TABLE, SET FLAG	INTERPRETER	A - Pg. 94
TIA TABLE, SET VARIABLE	INTERPRETER	A - Pg. 94
TRACE CORE	INTERPRETER	B - Pg. 99
TRACE TABLE	INTERPRETER	B - Pg. 101

INDEX

IBM Technical Newsletter

File Number S360-37

Re: Order No. GY28-6611-0

This Newsletter No. GY28-2371

Date November 15, 1968

Previous Newsletter Nos. None

IBM SYSTEM/360 OPERATING SYSTEM
TESTRAN
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of release 17 of the System/360 Operating System, provides replacement pages for the TESTRAN Program Logic Manual, Form Y28-6611-0. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

Cover,Preface
Contents
21-26,26.1
125,126
Comments, Back Cover

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

Modifications to the trace and go-back routines are necessary when operating on the Model 91. (Page 21, 23, 26.) The publication form number is changed in accordance with standards for program logic manuals.

File this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

IBM **Technical Newsletter**

File Number S/360-37 (OS Release 20.1)

Re: Order Number GY28-6611-0

This Newsletter Number GN26-8016

Date April 1, 1971

Previous Newsletter Numbers GY28-2371

IBM SYSTEM/360 OPERATING SYSTEM
TESTRAN
PROGRAM LOGIC MANUAL

This Technical Newsletter, a part of release 20.1 of the System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Front Cover
Contents
15-16.2
25-28
47, 48
121, 122

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This Technical Newsletter modifies the publication to indicate a change to SVC 61 to support TSO (Time Sharing Option) test.

IBM Corporation, Programming Publications, Dept. D78, San Jose, Calif. 95114

PRINTED IN U.S.A.

IBM **Technical Newsletter**

File Number S/360-37 (OS Release 20.1)

Re: Order Number GY28-6611-0

This Newsletter Number GN26-8018

Date September 1, 1971

Previous Newsletter Numbers GY28-2371
GN26-8016

IBM SYSTEM/360 OPERATING SYSTEM
TESTRAN
PROGRAM LOGIC MANUAL

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

25-26.1

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This Technical Newsletter replaces information inadvertently deleted in Technical Newsletter GN26-8016.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Publications, Dept. D78, San Jose, Calif. 95114

PRINTED IN U.S.A.

READER'S COMMENT FORM

IBM System/360 Operating System
TESTRAN
Program Logic Manual

GY28-6611-0
through TNL GN26-8018

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge
  - Other .....

- Please check the items that describe your position:
  - ☐ Customer personnel
  - ☐ IBM personnel
  - ☐ Manager
  - ☐ Systems Analyst
  - ☐ Operator
  - ☐ Programmer
  - ☐ Customer Engineer
  - ☐ Instructor
  - ☐ Sales Representative
  - ☐ Systems Engineer
  - ☐ Trainee
  - Other .....

- Please check specific criticism(s), give page number(s), and explain below:
  - ☐ Clarification on page(s) .....
  - ☐ Addition on page(s) .....
  - ☐ Deletion on page(s) .....
  - ☐ Error on page(s) .....

Explanation:

p. 11 - Reference to Control Program Services ?

p. 9, ¶2, 2nd + 3rd sentences imply that the the test save + open routines locate the object module; the 4th sentence implies that the problem program is executed following the edit.

p. 15-¶1, "exception" misspelled; Table 2: IEC038 should be IGC038

p 16 - ¶1, under SETUP ROUTINES; line 2: "fucntions" misspelled. ¶2, line 2: "If" should be "I"

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

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference 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. 81
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM S/360 Printed in U.S.A. GY28-6611-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

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