



**Systems Reference Library**

## **IBM System/360 Operating System**

### **TESTRAN**

TESTRAN is a facility for testing programs written in the assembler language for execution under the System/360 Operating System. It is intended for use by the individual programmer in testing his own programs.

This publication explains how to use TESTRAN for typical testing purposes, how to write essential job control statements, and how to interpret printed test results. It formally describes TESTRAN statements, cataloged procedures supplied by IBM, and TESTRAN diagnostic messages.

The information in this publication applies to systems that include the primary control program (PCP) or provide multiprogramming with a fixed number of tasks (MFT or Option 2). Amendments to this publication will supply information applicable to systems that provide multiprogramming with a variable number of tasks (MVT or Option 4).



First Edition (February 1967)

This publication is one of a set of three publications which entirely replace and obsolete the publications IBM System/360 Operating System: Data Management, Form C28-6537, and IBM System/360 Operating System: Control Program Services, Form C28-6541. The facilities and services available through the use of supervisor and data management macro-instructions are now described in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646. The descriptions and definitions of the supervisor and data management macro-instructions are contained in the publication IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647. The facilities available through the use of TESTRAN macro-instructions, as well as the descriptions and definitions of the TESTRAN macro-instructions, are contained herein.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

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

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

This publication describes the TESTRAN facility for testing programs written in the assembler language. It introduces this facility in Section 1, which shows by an example how TESTRAN helps in testing a program, and shows how the reader can use TESTRAN in testing his own programs.

Sections 2, 3, and 4 guide the reader in writing a source program, in writing job control statements, and in interpreting test results. The reader need not go beyond Section 2 before completing his source coding, and need not go beyond Section 3 before actually testing his program under the operating system. Also, he need not read any section in its entirety, because each treats a number of independent topics that can be referred to directly from the table of contents.

Several appendixes provide detailed descriptions of source statements, cataloged procedures, and diagnostic messages. Appendix A is of special interest, because it formally describes statements that are informally described in Section 2. The reader can use either Appendix A or Section 2 as the model for his own coding, depending on the style of presentation he prefers.

PREREQUISITE PUBLICATIONS

The following publications are prerequisites:

IBM System/360 Operating System: Introduction, Form C28-6534

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

IBM System/360 Operating System: Assembler Language, Form C28-6514

Knowledge of the macro-language, as described in the Assembler Language publication, is not required. However, the reader should know the general functions of system-defined macro-instructions (SAVE, OPEN, GET, PUT, DCB) that are introduced in the Concepts and Facilities publication and are fully described in the publications:

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647

PUBLICATIONS REFERRED TO IN THIS PUBLICATION

The following publications are referred to in this publication, but are not necessarily prerequisites:

IBM System/360 Operating System: Assembler (E) Programmer's Guide, Form C28-6595

IBM System/360 Operating System: Linkage Editor, Form C28-6538

IBM System/360 Operating System: Job Control Language, Form C28-6539

IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps, Form C28-6631



## CONTENTS

SECTION 1: INTRODUCTION. . . . .	8
Testing Procedure . . . . .	11
Requesting TESTSTRAN Services . . . . .	12
Structure of TESTSTRAN Statements. . . . .	12
Functions of TESTSTRAN Statements. . . . .	12
SECTION 2: HOW TO WRITE TESTSTRAN STATEMENTS . . . . .	14
Basic Recording Functions . . . . .	14
How to Dump a Storage Area . . . . .	14
How to Dump Changes to a Storage Area. . . . .	16
How to Dump a Dummy Control Section. . . . .	17
How to Dump Storage Maps, Registers, and Control Blocks. . . . .	20
How to Control Output Format . . . . .	22
How to Trace Control Flow and References to Data . . . . .	25
How to Comment the TESTSTRAN Listing . . . . .	29
How to Classify Test Information for Selective Retrieval . . . . .	29
Testing of Complex Programs . . . . .	30
How to Test a Module Already in a Library. . . . .	30
How to Enlarge on a Partially Tested Program . . . . .	32
How to Test an Overlay Program . . . . .	33
How to Test a Dynamic Serial Program . . . . .	34
SECTION 3: HOW TO WRITE JOB CONTROL STATEMENTS . . . . .	35
Assembly. . . . .	35
Linkage Editing . . . . .	36
Execution . . . . .	37
TESTSTRAN Editing . . . . .	38
Assembly and Linkage Editing. . . . .	39
Assembly, Linkage Editing, and Execution. . . . .	41
Assembly, Linkage Editing, Execution, and TESTSTRAN Editing . . . . .	44
SECTION 4: HOW TO INTERPRET SYSTEM OUTPUT. . . . .	49
Page Heading (...TESTSTRAN OUTPUT...) . . . . .	52
Test Point Identification (AT LOCATION...). . . . .	52
Statement Output (...MACRO ID...) . . . . .	52
DUMP CHANGES Output. . . . .	53
DUMP COMMENT Output. . . . .	53
DUMP DATA Output . . . . .	53
DUMP MAP Output. . . . .	54
DUMP PANEL Output. . . . .	55
DUMP TABLE Output. . . . .	56
ERROR Message. . . . .	56
TEST CLOSE Output. . . . .	57
TEST OPEN Output . . . . .	57
TRACE CALL Output. . . . .	58

TRACE FLOW Output. . . . .	59
TRACE REFER Output . . . . .	60
TRACE STOP Output. . . . .	61
TESTRAN Statement Trace (EXECUTED STATEMENTS...).	62
TESTRAN Editor Message (** IEGE...)	62
APPENDIX A: FORMAL DESCRIPTION OF TESTRAN STATEMENTS.	63
Coding Conventions. . . . .	63
Functions of TESTRAN Statements . . . . .	63
DUMP and TRACE Statements. . . . .	64
TEST Statements. . . . .	66
Linkage Statements. . . . .	66
Specification Statements. . . . .	67
Decision-Making Statements. . . . .	68
GO Statements. . . . .	69
SET Statements . . . . .	70
Format of TESTRAN Statements. . . . .	71
APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES . . . . .	73
Procedure ASMEC . . . . .	73
Procedure LKED. . . . .	73
Procedure TASME . . . . .	73
Procedure TASMEG. . . . .	74
Procedure TASMEGED. . . . .	74
Procedure TTED. . . . .	74
APPENDIX C: TESTRAN MESSAGES . . . . .	75
TESTRAN Editor Messages . . . . .	76
TESTRAN Interpreter Messages. . . . .	77
TESTRAN Macro-Expansion Messages. . . . .	85
INDEX . . . . .	89

FIGURES

Figure 1.	Use of TESTRAN to Detect an Error in a Program. . . . .	9
Figure 2.	Combination of TESTRAN and Problem Program Source Modules. . . . .	11
Figure 3.	Execution Time Testing of the Problem Program . . . . .	11
Figure 4.	Printing of Test Information. . . . .	12
Figure 5.	Job Control Statements for Assembly . . . . .	35
Figure 6.	Job Control Statements for Linkage Editing. . . . .	36
Figure 7.	Job Control Statements for Execution. . . . .	37
Figure 8.	Job Control Statements for TESTRAN Editing. . . . .	38
Figure 9.	Job Control Statements for Assembly and Linkage Editing .	39
Figure 10.	Job Control Statements for Assembly, Linkage Editing, and Execution. . . . .	42
Figure 11.	Job Control Statements for Assembly, Linkage Editing, Execution, and TESTRAN Editing . . . . .	45
Figure 12.	TESTRAN Editor Listing: Sample Page . . . . .	50

TABLES

Table 1.	Printing Formats for Data Types. . . . .	51
Table 2.	Format of TESTRAN Statements . . . . .	71
Table 3.	Definitions of Abbreviations Used in Table 2 . . . . .	71
Table 4.	Definitions of Variables Used in Tables 2 and 3. . . . .	71
Table 5.	Definition of Type, Length, and Scale. . . . .	71
Table 6.	TESTRAN Messages . . . . .	75

## SECTION 1: INTRODUCTION

The testing of a major program can be as time-consuming as the design and coding of its routines. Although testing is always time well spent, the need to meet deadlines often leads to incomplete testing and subsequent failures. And a failure in a single control section can delay an entire project.

To help in testing programs, the IBM System/360 Operating System offers a facility known as the test translator, or TESTRAN. This facility helps to uncover faulty logic by providing printed information about the actual working of a program. At the programmer's direction, TESTRAN describes the changing contents of storage areas, registers, and control blocks, and also the way in which control flows from one group of instructions to another.

As an example, the test of a subroutine named PRIMER is shown in Figure 1. For any positive number X, PRIMER is designed to find the smallest number greater than X that is a prime number. The TESTRAN listing shows that PRIMER contains an error, because, as shown at (1) in the figure, it returns a result of 3 rather than 2 for X = 1.

From the TESTRAN listing, the programmer can reconstruct the flow of data and control that occurred during execution of PRIMER. As shown at (2), the value X = 1 was loaded into general register 10 before execution of the instruction assembled at 000064. Branches were made to ODD and GOT. The erroneous result +3 was stored from general register 11 before execution of the RETURN macro-instruction assembled at 0000C0.

Tracing the flow of control, it is easy to find the instructions that caused the error. Because X was an odd value, it was moved to register 11 and, at (3), increased by two. The result, being a prime number, was stored as the answer. The error is obviously based on the assumption that, if X is an odd number, the next larger prime number must also be an odd number. In the single case X=1, the assumption is invalid.

The error in PRIMER is simple enough that it might easily be recognized even without the help of TESTRAN. From this example, however, it should be clear that TESTRAN could be most helpful in finding hidden and complicated errors. In addition, one should remember that even so trivial an error could be difficult to find if the subroutine were part of a large, complex program.

A TESTRAN listing, such as that shown in Figure 1, is printed after execution of the program being tested. During execution, TESTRAN can provide an additional service by checking for predefined error conditions and taking corrective actions when necessary. For example, the programmer might know that some value in his program should never exceed a certain maximum. The value might be a result computed by a subroutine, or it could be a counter used to control a processing loop. TESTRAN could be used to check the value and, if the maximum were exceeded, to substitute a lesser value or to pass control to some other part of the program. Of course, the final results of the program would probably be incorrect, but the continued processing would offer the chance of finding other errors not related to the faulty loop or subroutine.

# TESTRAN LISTING

ASSEMBLY

1

For the value X = +1, the result returned by PRIMER is +3. It should be +2.

TESTING TESTRAN OUTPUT DATE 66/084 TIME 00/00 PAGE 5

1) MACRO ID 003, DUMP CHANGES

NONE

1) MACRO ID 002, TRACE FLOW , TTPRIME , FROM (PRIMER ) 000064 005784, CC=0  
SVC 26 G'00' 0000003C G'01' 8000581C G'14' 4000582A G'15' 00005778

AT LOCATION (PRIMER ) 000064 005784 ENTER TTPRIME  
1) MACRO ID 006, DUMP PANEL  
G'10' +1  
PSW FF 0 5 0026 4 0 005786 CC=0 FIX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF

1) MACRO ID 002, TRACE FLOW , TTPRIME , FROM (PRIMER ) 000072 005792 TO ODD (PRIMER ) 000082 0057A2, CC=1  
BC 70 F 02A G'15' 00005778

1) MACRO ID 002, TRACE FLOW , TTPRIME , FROM (PRIMER ) 000094 0057B4 TO GOT (PRIMER ) 0000B8 0057D8, CC=2  
BC 20 F 060 G'15' 00005778

1) MACRO ID 002, TRACE FLOW , TTPRIME , FROM (PRIMER ) 0000C0 0057E0, CC=2  
SVC 26 G'00' 0000003C G'01' 8000581C G'14' 4000582A G'15' 00005778

AT LOCATION (PRIMER ) 0000C0 0057E0 ENTER TTPRIME  
1) MACRO ID 009, TRACE STOP TTPRIME 002  
1) MACRO ID 010, DUMP PANEL  
G'09' +3 G'10' +0 G'11' +3  
PSW FF 0 5 0026 6 0 0057E2 CC=2 FIX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF

AT LOCATION TTSVC2 (CALLTEST) 0000EA 00582A ENTER DATAGEN

1) MACRO ID 010, DUMP DATA STARTING IN SECTION CALLTEST

00FB	X	RESULT
0C5830	+1	+3

AT LOCATION (PRIMER ) 000058 005778 ENTER TTPRIME

2

The value X=+1 was loaded into general register 10 (G'10').

A branch was made to ODD from relative location 000072.

A branch was made to GOT from relative location 000094.

A result of +3 was stored from general register 11 (G'11').

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
000058				150	PRIMER CSECT
000058				151	USING *,15
				152	SAVE (14,12)
00005C	58C1 0000		00000	155	L 12,0(1)
000060	58AC 0080		00000	156	L 10,0(12)
000064	12AA			157	LTR 10,10
000066	47C0 F072		000CA	158	BC 12,ERR
00006A	1BBB			159	SR 11,11
00006C	8CA0 0001		00001	160	SRDL 10,1
000070	12BB			161	LTR 11,11
000072	4770 F02A		00082	162	BNZ ODD
000076	8CAB 001F		0001F	163	SRDL 10,31
00007A	4AB0 F080		00080	164	AH 11,=H'1'
00007E	47FC F032		0008A	165	B LOAD
000082	8CA0 001F		0001F	166	ODD SRDL 10,31
000086	4AB0 F082		000DA	167	AH 11,=H'2'
00008A	4190 0003		00003	168	LOAD LA 9,3
00008E	1859			169	AGAIN LR 5,9
000090	1C45			170	MR 4,5
000092	195B			171	CR 5,11
000094	4720 F060		00088	172	BH GOT
000098	4780 F058		00080	173	BE INCR
00009C	183B			174	LR 3,11
00009E	1B22			175	SR 2,2
0000A0	1D20			176	DR 2,9
0000A2	1222			177	LTR 2,2
0000A4	4780 F052		00080	178	BZ INCR
0000A8	4A90 F082		000DA	179	AH 9,=H'2'
0000AC	47F0 F036		0008E	180	B AGAIN
0000B0	4ABC F082		000DA	181	INCR AH 11,=H'2'
0000B4	47F0 F032		0008A	182	B LOAD
0000B8	58C1 0004		00004	183	GOT L 12,4(1)
0000BC	50BC 0000		00000	184	ST 11,0(12)
(0000C0)				185	RETURN (14,12),
				189	ERR RETURN (14,12),
0000D8				194	LTORG
0000D8	0001			195	=H'1'
0000DA	0002			196	=H'2'
000058				197	END

TESTRAN LISTING

ASSEMBLY LISTING

RAN OUTPUT DATE 66/084 TIME 00/00 PAGE 5

ME , FROM (PRIMER ) 000064 005784, CC=0  
'01' 8000581C G'14' 4000582A G'15' 00005778

ENTER TTPRIME  
IX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF

ME , FROM (PRIMER ) 000072 005792 TO ODD (PRIMER ) 000082 0057A2, CC=1

ME , FROM (PRIMER ) 000094 0057B4 TO GOT (PRIMER ) 0000B8 0057D8, CC=2

ME , FROM (PRIMER ) 0000C0 0057E0, CC=2  
'01' 8000581C G'14' 4000582A G'15' 00005778

ENTER TTPRIME  
002  
IX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF

A 00582A ENTER DATAGEN

3 IN SECTION CALLTEST

ENTER TTPRIME

2

The value X=1 was loaded into general register 10 (G'10').

A branch was made to ODD from relative location 000072.

A branch was made to GOT from relative location 000094.

A result of +3 was stored from general register 11 (G'11').

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
000058				150	PRIMER CSECT
000058				151	USING *,15
				152	SAVE (14,12)
00005C	58C1 0000		00000	155	L 12,0(1)
000060	58AC 0000		00000	156	L 10,0(12)
000064	12AA			157	LTR 10,10
000066	47C0 F072		000CA	158	BC 12,ERR
00006A	1BBE			159	SR 11,11
00006C	8CA0 0001		00001	160	SRDL 10,1
000070	12BB			161	LTR 11,11
000072	4770 F02A		00082	162	BNZ ODD
000076	8CA0 001F		0001F	163	SRDL 10,31
00007A	4AB0 F080		00008	164	AH 11,=H'1'
00007E	47FC F032		0000A	165	B LOAD
000082	8CA0 001F		0001F	166	ODD SRDL 10,31
000086	4AB0 F082		000DA	167	AH 11,=H'2'
00008A	4190 0003		00003	168	LOAD LA 9,3
00008E	1859			169	AGAIN LR 5,9
000090	1C45			170	MR 4,5
000092	195B			171	CR 5,11
000094	4720 F060		00088	172	BH GOT
000098	4780 F058		00080	173	BE INCR
00009C	183B			174	LR 3,11
00009E	1822			175	SR 2,2
0000A0	1D29			176	DR 2,9
0000A2	1222			177	LTR 2,2
0000A4	4780 F058		00080	178	BZ INCR
0000A8	4A90 F082		000DA	179	AH 9,=H'2'
0000AC	47F0 F036		0008E	180	B AGAIN
0000B0	4ABC F082		000DA	181	INCR AH 11,=H'2'
0000B4	47F0 F032		0008A	182	B LOAD
0000B8	58C1 0000		00004	183	GOT L 12,4(1)
0000BC	508C 0000		00000	184	ST 11,0(12)
(0000C0)				185	RETURN (14,12),RC=4
				189	ERR RETURN (14,12),RC=0
0000D8				194	LTORG
0000D8	0001			195	=H'1'
0000DA	0002			196	=H'2'
000058				197	END

E 01FEB66 3/30/66

3

Because X was odd, X+2 was the first tentative result. The logic errs, because X+1 is the correct result when X=1.

Figure 1. Use of TESTRAN to Detect an Error in a Program

TESTING PROCEDURE

Requests for TESTRAN services are coded in a TESTRAN source module. This module is combined with the program to be tested (the problem program) either by the assembler or by the linkage editor, as shown in Figure 2. In the first case, the TESTRAN and problem program source modules are assembled together and result in a single object module. In the second case, the source modules are assembled separately, result in separate object modules, and are processed by the linkage editor to form a single load module.

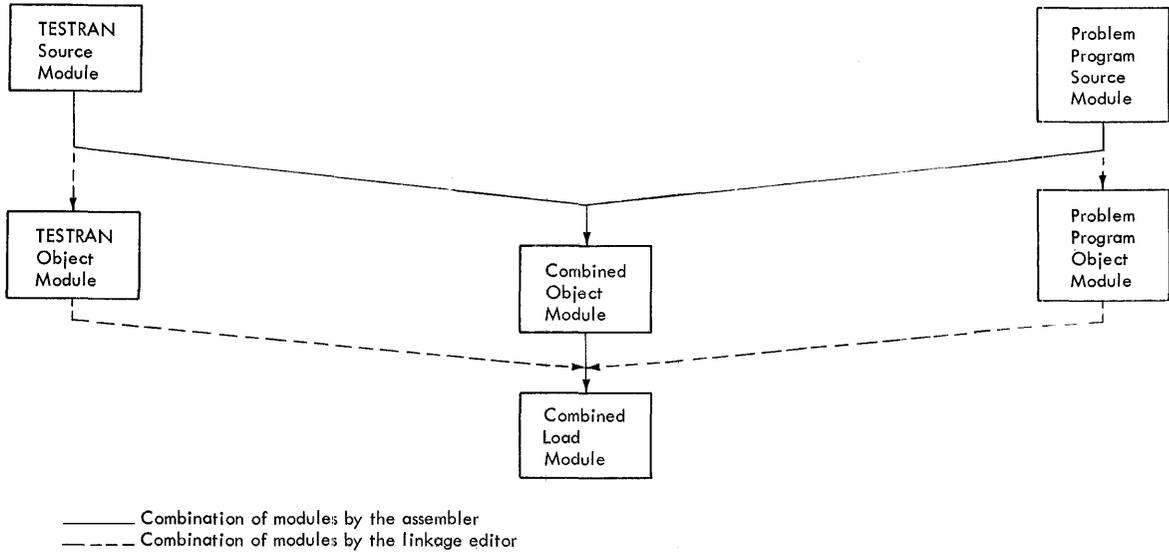


Figure 2. Combination of TESTRAN and Problem Program Source Modules

The single load module is loaded and executed as a problem program. Requests for test services are interpreted by the TESTRAN interpreter, a component of the control program that receives control during program interruptions. As shown in Figure 3, the TESTRAN interpreter places test information in a TESTRAN data set, along with control information which it copies from the unloaded form of the load module.

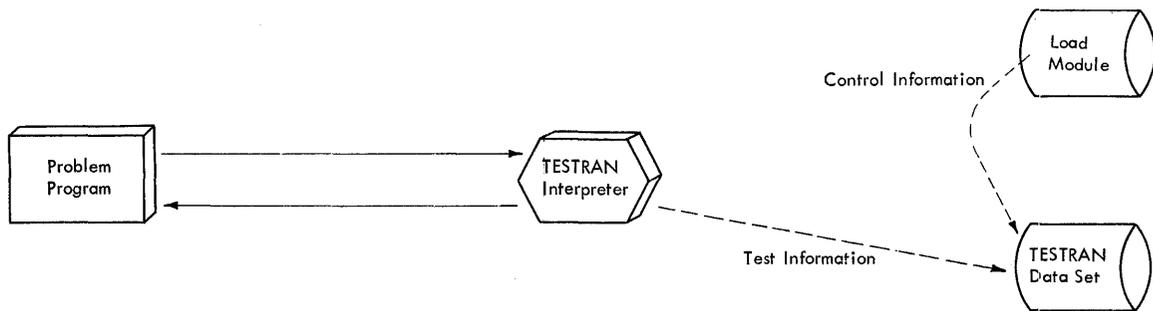


Figure 3. Execution Time Testing of the Problem Program

Test information, in the form of dumps and traces, is printed by the TESTRAN editor, as shown in Figure 4. A dump is a symbolic representation of data as it existed at a particular time during execution of the problem program. A trace is a record of control flow or references to data over a period of time.



Figure 4. Printing of Test Information

Like the assembler and the linkage editor, the TESTSTRAN editor is a processing program that is executed as a job step. It uses the control information copied from the load module to edit test information into a meaningful symbolic format. The control information includes symbol tables and a control dictionary for each object module that is included in the load module. The control dictionary is produced as a standard feature of assembly, while the symbol table is produced as an optional feature. Both are placed in the load module as an optional feature of linkage editing.

#### REQUESTING TESTSTRAN SERVICES

Requests for TESTSTRAN services are written as statements in the TESTSTRAN source module. Each statement is a coded TESTSTRAN macro-instruction, which the assembler automatically replaces with a series of constants. The constants, in effect, are a control statement that directs the TESTSTRAN interpreter to perform a specific operation.

When the interpreter performs a requested operation, the operation itself determines whether the next sequential macro-instruction is interpreted, or whether a logical branch is made to some other macro-instruction. The process of interpreting a TESTSTRAN macro-instruction thus resembles the execution of a machine instruction, and is more conveniently referred to hereafter as the execution of a TESTSTRAN statement.

#### STRUCTURE OF TESTSTRAN STATEMENTS

The structure of TESTSTRAN statements is similar to that of statements in the basic assembler language. Each statement includes an operation code and one or more operands. The operation code can be preceded by a symbolic name, and the operands can be followed by a comment.

The operation code and first operand together define the type of operation to be performed, and are used as generic names for statements. For example, a DUMP MAP statement dumps a map of control sections and allocated storage areas; the operand MAP distinguishes this statement from DUMP statements that request other types of dump operations.

#### FUNCTIONS OF TESTSTRAN STATEMENTS

The operations requested by TESTSTRAN statements provide the following general functions:

- Recording functions, which provide dumps and traces of the problem program.
- Linkage functions, which control linkage to the TESTSTRAN interpreter.

- Decision-making functions, which provide condition testing and conditional branching.
- Branching functions, which provide unconditional branching and subroutine capabilities.
- Assignment functions, which control values of variables in the problem program and of special variables used in decision making.

These functions are provided by statements that are formally described in Appendix A. Functional descriptions of the statements appear in the next section, which describes how to write statements for typical test applications.

## SECTION 2: HOW TO WRITE TESTRAN STATEMENTS

This section shows how to write TESTRAN statements to perform typical testing functions. It gives examples of statements for performing each function, and the reader can adapt these examples to his own needs. If there is some question about adapting a specific example, refer to Appendix A for complete, formal descriptions of the statements involved.

Section 2 has two parts:

- Basic Recording Functions.
- Testing of Complex Programs.

The first part shows how to program various types of dumps and traces. The second part shows how to test programs that are not simply structured or not formed from single object modules.

The first part of this section should be of general interest, while the second should be read or ignored according to individual need. Each part discusses various topics, and these also should be studied in a selective fashion.

### BASIC RECORDING FUNCTIONS

This part of Section 2 describes various types of dumps and traces. Remember that a dump represents data as it exists at a particular time; a trace represents control flow or references to data over an extended period of time.

#### HOW TO DUMP A STORAGE AREA

Assume that the program containing the area is very simple and can be represented as follows:

```
ENTRY    SAVE    (14,12)
        .
        .
PROCESS  MVC     MYDATA(20),0(6)
        .
        .
MYDATA   DC     C'DATAAREA'
        DC     F'0,1,2'
        .
        .
        END     ENTRY
```

The problem might then be to dump the 20-byte area beginning at MYDATA, just before the contents are changed by PROCESS. If so, the next listing shows a solution:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
          DUMP DATA,MYDATA,MYDATA+20
ENTRY     SAVE (14,12)
          .
          .
          .
PROCESS   MVC MYDATA(20),0(6)
          .
          .
          .
MYDATA    DC C'DATAAREA'
          DC F'0,1,2'
          .
          .
          .
          END NEWENTRY

```

Execution begins at NEWENTRY, the beginning of a TESTRAN sequence that means "Enter the problem program at ENTRY; at PROCESS, dump the area from MYDATA to MYDATA+20." In this sequence, only the first statement is actually executed. This statement uses the information in another statement (TEST AT) to synchronize testing specified by a third statement (DUMP DATA) with execution of the problem program. It establishes a test point (a special link to the TESTRAN interpreter) at PROCESS, and passes control to ENTRY. When PROCESS is reached, the interpreter executes the DUMP DATA statement; it returns control to the problem program, where the MVC instruction is executed. The dump is printed as:

```

0090 MYDATA
005F68 DATAAREA +0 +1 +2

```

assuming that MYDATA was assembled at location 000090 and loaded at location 005F68.

To dump more than one area, the programmer simply writes additional DUMP DATA statements:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
          DUMP DATA,MYDATA,MYDATA+20
          DUMP DATA,0(0,6),80(0,6)
          .
          .
          .

```

To dump these areas at more than one point in the program, he specifies additional instruction addresses in the TEST AT statement:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,(PROCESS,INPUT,INPUT+18)
          .
          .
          .

```

To dump different areas at various test points, he uses additional TEST AT statements:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
          DUMP DATA,MYDATA,MYDATA+20
          TEST AT,(INPUT,INPUT+18)
          DUMP DATA,0(0,6),80(0,6)
          DUMP DATA,TABLE(4),TABLE+8(4)
          .
          .
          .

```

#### HOW TO DUMP CHANGES TO A STORAGE AREA

The method is the same as for dumping a storage area; the basic difference is that CHANGES replaces DATA in the DUMP statement:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
          DUMP CHANGES,MYDATA,MYDATA+20
ENTRY     SAVE (14,12)
          .
          .
          .
PROCESS   MVC MYDATA(20),0(6)
          .
          .
          .
MYDATA    DC C'DATAAREA'
          DC F'0,1,2'
          .
          .
          .
          END NEWENTRY

```

Execution begins at NEWENTRY and continues at ENTRY. Before PROCESS is executed, the TESTRAN interpreter dumps the 20 byte area at MYDATA. If PROCESS is executed three times, the dumps may appear as:

```

0090 MYDATA
005F68 DATAAREA +0 +1 +2
-----
0090 MYDATA
005F68 WORKAREA +3
00A0
005F78 -40
-----
00A0
005F78 -41

```

The first dump shows the full contents of the four fields assembled at 000090 and loaded at 005F68. The second shows changes to the first, second, and fourth fields, and shows that the third field is unchanged. The third dump shows that only the fourth field has changed since the previous dump.

To show changes to an area, a DUMP CHANGES statement must be executed more than once. If PROCESS were executed only once, the example would have to be changed to specify additional test points:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,(PROCESS,INPUT,INPUT+18)
          DUMP CHANGES,MYDATA,MYDATA+20
          .
          .
          .

```

Change dumps would then occur at the test points PROCESS, INPUT, and INPUT+18. There might be other TESTRAN statements to be executed, however, and these statements might not be the same for each test point. In this case, it would be necessary to use branching statements:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
CONTINUE DUMP CHANGES,MYDATA,MYDATA+20
          TEST AT,INPUT
          DUMP DATA,0(0,6),80(0,6)
          GO TO,CONTINUE
          TEST AT,INPUT+18
          DUMP DATA,TABLE(4),TABLE+8(4)
          GO TO,CONTINUE
          .
          .
          .

```

The statement CONTINUE is the last executed at each test point. The GO TO statements in no way affect the logic of the program being tested; control is returned to each test point in the normal manner.

To dump changes to more than one area of storage, the programmer should specify each area in a separate statement:

```

NEWENTRY TEST OPEN,ENTRY
          TEST AT,PROCESS
          DUMP CHANGES,MYDATA,MYDATA+20
          DUMP CHANGES,TABLE(4),TABLE+8(4)
          .
          .
          .

```

Each statement produces a separate series of change dumps, even if two statements should specify the same storage area. Each dump shows changes to the area since the last dump by the same statement.

Changes in index values redefine areas that are specified by indexed addresses. For example, the statement

```
DUMP CHANGES,ALPHA(4),ALPHA+60(4)
```

dumps a 60-byte area whose location depends on an index value in general register 4. On the first execution of the statement, the index value might be zero, causing a dump of the area from ALPHA to ALPHA+60. On the next execution, the index value might be 40, redefining the dumped area as that from ALPHA+40 to ALPHA+100. The second dump would show changed fields from ALPHA+40 to ALPHA+60 and all fields from ALPHA+60 to ALPHA+100.

#### HOW TO DUMP A DUMMY CONTROL SECTION

A dummy control section describes a storage area without actually reserving the area. The area may be allocated during execution, or may be reserved by a regular control section, as in the following example:

```

ENTRY    SAVE    (14,12)
        .
        .
        .
        LA      4,MYDATA
        USING  DUMMY,4
PROCESS  MVC     DUMMY(20),0(6)
        .
        .
        .
MYDATA   DC     C'DATAAREA'
        DC     F'0,1,2'
        .
        .
        .
DUMMY    DSECT
COUNT  DS     XL4
NUMBERS DS     4F
        END    ENTRY

```

This program defines a dummy control section named DUMMY, and assigns it the storage reserved for MYDATA. The example otherwise is the same as that used in "How to Dump a Storage Area." The instruction named PROCESS here refers to DUMMY rather than MYDATA, but its effect is the same as in the earlier example.

Assume that DUMMY is to be dumped after PROCESS has been executed, and that the 20-byte area at MYDATA is to be dumped as before. The program then becomes:

```

NEWENTRY TEST  OPEN,ENTRY
        TEST  AT,PROCESS
        DUMP  DATA,MYDATA,MYDATA+20
        TEST  AT,PROCESS+6
        USING DUMMY,4
        DUMP  DATA,COUNT,NUMBERS+16,DSECT=DUMMY
ENTRY    SAVE    (14,12)
        .
        .
        .
        LA      4,MYDATA
        USING  DUMMY,4
PROCESS  MVC     DUMMY(20),0(6)
        .
        .
        .
MYDATA   DC     C'DATAAREA'
        DC     F'0,1,2'
        .
        .
        .
DUMMY    DSECT
COUNT  DS     XL4
NUMBERS DS     4F
        END    NEWENTRY

```

As before, execution begins at NEWENTRY, control is passed to ENTRY, and the area of MYDATA is dumped at PROCESS. After PROCESS is executed, the new statements dump the 20 bytes from COUNT to NUMBERS+16. Thus, the two dumps of the same area might appear as follows:

```

0090 MYDATA
005F68 DATA AREA +0 +1 +2

0000 COUNT NUMBERS
005F68 00002A6 -647 +30 -1

```

The dumps show that MYDATA was assembled at 000090 and that COUNT was assembled at 000000; both had the same location (005F68) when dumped.

Note that a special operand (DSECT=DUMMY) points to a dummy control section, which is made addressable by a USING statement. A USING statement is not needed preceding the other TESTRAN statements, since their address operands are assembled as A-type address constants.

A dummy control section may describe more than one area of storage; for example, it may define each of several buffers in a buffer pool. If the areas are contiguous, they can be dumped by a single statement, as in the following example:

```

NEWENTRY TEST OPEN, ENTRY
          TEST AT, PROCESS+6
          USING DUMMY, 4
          DUMP DATA, COUNT, NUMBERS+16, DSECT=(DUMMY, 3)
ENTRY    SAVE (14, 12)
          .
          .
          LA 4, MYDATA
          USING DUMMY, 4
PROCESS  MVC DUMMY(60), 0(6)
          .
          .
MYDATA  DC C'DATAAREA'
          DC F'0,1,2'
          DS XL40
          .
          .
DUMMY   DSECT
COUNT  DS XL4
NUMBERS DS 4F
          END NEWENTRY

```

PROCESS moves data into a 60-byte area beginning at DUMMY, i.e., at MYDATA. This area is dumped as three 20-byte areas ((NUMBERS+16)-COUNT=20), each area having the format defined in DUMMY:

```

0000 COUNT NUMBERS
005F68 00002A6 -647 +30 -1

0000 COUNT NUMBERS
005F7C 00000006 +4 +0 -2

0000 COUNT NUMBERS
005F90 000001CF +278 -64 -89

```

Changes to a dummy control section can be dumped, just as changes to a regular control section. For this purpose, a DUMP CHANGES statement (with a DSECT operand) is used in place of a DUMP DATA statement. For examples of the use of DUMP CHANGES, refer to "How to Dump Changes to a Storage Area."

## HOW TO DUMP STORAGE MAPS, REGISTERS, AND CONTROL BLOCKS

For simplicity, assume that a storage map, registers, and control blocks should all be dumped at X in the following program:

```

START    SAVE    (14,12)
        .
        .
        .
        OPEN    (MYDCB,(OUTPUT))
X
        .
        .
        .
MYDCB    DCB     DSORG=PS,MACRF=(PM),DDNAME=MYDD
        END     START
    
```

The unshaded statements below perform these functions:

```

NEWSTART TEST OPEN,START
        TEST AT,X
        DUMP MAP
        DUMP PANEL
        DUMP TABLE,TCB
        DUMP TABLE,DCB,MYDCB
        DUMP TABLE,DEB,MYDCB
START    SAVE    (14,12)
        .
        .
        .
        OPEN    (MYDCB,(OUTPUT))
X
        .
        .
        .
MYDCB    DCB     DSORG=PS,MACRF=(PM),DDNAME=MYDD
        END     NEWSTART
    
```

Execution begins at NEWSTART, where X is established as a test point. Control passes to START, and the DUMP statements are executed at X. The dumps appear as follows:

### Storage Map (recorded by DUMP MAP):

NAME	TYPE	CSECT NAME	ASSEMBLED AT	LOADED AT	LENGTH-DEC	HEX
GO	LOADED PROGRAM	NEWSTART	000000	009020	47	2F
	LOADED PROGRAM		000030	009050	172	AC
IEGTRNK	LOADED PROGRAM			009120	1048	418
IEGTTROT	LOADED PROGRAM			07F3D0	1160	488
	OBTAINED STORAGE			07F858	96	60
	OBTAINED STORAGE			07F948	560	230
	OBTAINED STORAGE			07F8B0	360	168

### Registers (recorded by DUMP PANEL):

```

,
G'00' 0007FD58 G'01' 0007FD58 G'02' 00000058 G'03' 50009050 G'04' 00006EE8 G'05' 0007FF5C G'06' 000054B0 G'07' 00000000
G'08' 0000003C G'09' 40011062 G'10' 0007FF1C G'11' 0007FF5C G'12' 00000180 G'13' 0007FE98 G'14' 50009088 G'15' 92007750
PSW FF 1 5 0026 4 0 00908A CC=0 FIX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF
, F'0' 00000000 00000000 F'2' 00000000 00000000 F'4' 00000000 00000000 F'6' 00000000 00000000
,
    
```

Task Control Block (recorded by DUMP TABLE,TCB):

SECTION	FIELD NAME	CONTENTS
	TCBFRS	00000000 00000000 00000000 82000170 00040000 0007DCB8 00000000 00000000
	TCBRBP	00009100
	TCBPPIE	00000000
	TCBDEB	0007FCDC
	TCBTIO	0007FF5C
	TCBCMP	00000000
	TCBTRN	0007F948
	TCBMSS	00005670
	TCBPKF	10
	TCBFLGS	00000000 00000000 00000000 00000000 00000000
	TCBLMP	000
	TCBDSP	000
	TCBLLS	0007F3A8
	TCBJLB	00000000
	TCBJSE	00000000
	TCBGRS	000000C6 000054B0 800092F4 0007FB04 4007FF844 50004C1A 00000001 0007FAF0 0007FA90 0007FE58
		0007FAF0 04000030 010000AC 40404040 40404040 40404040
	TCBIDF	01000000
	TCBFSA	404040
	TCBTCB	40404040
	TCBTME	40404040

Data Control Block (recorded by DUMP TABLE,DCB,MYDCB):

SECTION	FIELD NAME	CONTENTS
DEVICE DEPENDENT INTERFACES		
	DCB	00000000 00000000 00000000 00000001 00810000
COMMON INTERFACE		
	DCB	0207FC10 00004000 00000001
FOUNDATION BLOCK EXTENSION		
	DCB	42000001 80000000
FOUNDATION BLOCK		
	DCB	00400050 0007FCDC 92
ACCESS METHOD INTERFACES		
	DCB	00775000 007B880C 00000100 09005028 28282840 07FB0000 07FCB800 07FCB800 00005000 00000100
		00000000 00884848 70201EC9 C5C7E3E3 D9D5C600 4C0040F6 40404000 00000000 00000002 0C26FE06
		78000140 40404040 40404000 00000000 00000002 0027FE06 78000140 40404040 40404000 00000000
		00000002 0028FE06 78000140 40404040 40404000 00000000 00000002 0029FE06 18000140 4C404040
		40404000 00000000 00000002 002AFE06 78000340 40404040 40404000 00000000 00000002 0C2DFE06
		78000106

Data Extent Block (recorded by DUMP TABLE,DEB,MYDCB):

SECTION	FIELD NAME	CONTENTS
PREFIX SECTION		
	DEBKARA	00
	DEBDCBA	00000000 000000
	DEBDCBMK	00000000 00000001 10011111 11100000
	DEBLNGTH	0C
NUCLEUS		
	DEBNMSUB	003
	DEBTCBAD	000180
	DEBAMLNG	004
	DEBDEBAD	07F87C
	DEBOFLGS	11001000
	DEBIRBAD	000000
	DEBOPATB	00001111
	DEBSYSPG	000000
	DEBNMEXT	001
	DEBUSRPG	000000
	DEBPRIOR	000
	DEBECBAD	000000
	DEBPROTG	001
	DEBDEBID	015
	DEBDCBAD	00909C
	DEBXSCL	002
	DEBAPPAD	07FCB8
EXTENT		
	DEB	33002000
ACCESS METHOD		
	DEB	00010000
SUBROUTINE ID		
	DEB	C1D9C1D2 1000

The format of each dump is explained in "Section 4: How to Interpret System Output." Note that:

- The storage map shows the length and location of each program that was loaded and each storage area that was obtained for the active task (job step). The first program (GO) is the problem program; the others are components of the TESTRAN interpreter. GO includes two control sections: NEWSTART, which is defined by the TEST OPEN statement and contains all five TESTRAN statements, and an unnamed control section, which contains the problem program instructions.
- The dump of registers includes both the general and floating-point registers, assuming that the computing system includes the floating-point option. It also includes the program status word (PSW) that was stored when the problem program was interrupted at the current test point.
- The dumps of control blocks show the task control block (TCB) for the active task (job step), the data control block (DCB) named MYDCB, and the data extent block (DEB) created during the opening of MYDCB.

In Figure 5, the contents of all registers appear in hexadecimal format. The programmer can specify a different format (such as fixed-point or floating-point) in the DUMP PANEL statement (refer to "How to Control Output Format). Since the specified format applies to all registers dumped by the statement, it is often desirable to use separate statements for dumping general and floating-point registers:

```
DUMP PANEL,G'0,15'  
DUMP PANEL,F'0,6'
```

The first statement dumps the general registers 0 to 15; the second dumps the floating-point registers 0 to 6. The programmer can also select specific registers, as in the statement

```
DUMP PANEL,(G'4',G'SUM',G'8,9',G'13,1')
```

which dumps only the following general registers:

- Register 4.
- The register whose number is the value of the symbol SUM.
- Registers 8 and 9.
- Registers 13, 14, 15, 0, and 1.

Of course, if the programmer wishes to dump specific general and floating-point registers, and to dump both in the same format, he can specify them in a single statement, such as:

```
DUMP PANEL,(G'5',F'SUM',F'4,6',G'8,10')
```

#### HOW TO CONTROL OUTPUT FORMAT

The TESTRAN editor determines the format of the output from most TESTRAN statements. However, the statements

```
DUMP DATA  
DUMP CHANGES  
DUMP PANEL  
TRACE REFER
```

produce output whose format may be determined in any of three ways:

1. By special operands.
2. By symbol tables.
3. By default.

By understanding each way of determining format, and the conditions under which it is used, the programmer can control the format of data recorded from registers and main storage.

**SPECIAL OPERANDS:** There are two operands by which the programmer can specify output format:

- The DATAM operand, which defines storage field or register format.
- The NAME operand, which defines a field name.

The DATAM operand can be used in any of the four statements; the NAME operand can be used in a DUMP DATA or DUMP CHANGES statement.

The DATAM Operand: The DATAM operand specifies the format of a field or register in terms of three attributes:

- Type
- Length
- Scale

The specification of attributes is similar to that in an assembler DC or DS statement and is illustrated by the following statements:

```
D1      DUMP  DATA, INPUT+6, DATAM=L74
D2      DUMP  CHANGES, 0(0,13), 72(0,13), DATAM=L4
D3      DUMP  PANEL, F'0,6', DATAM=D
T1      TRACE REFER, TABLE, TABLE+80, DATAM=FL4S-2
```

D1 dumps a single field that begins at INPUT+6. The length of the field is 74 bytes; because no type is specified, the contents of the field are printed as hexadecimal data.

D2 dumps a series of up to eighteen 4-byte fields, each containing changes to the contents of a 72-byte storage area.

D3 dumps the old program status word (OPSW) and the contents of the floating-point registers. The type of data in the registers is specified as D (long floating-point), which implies a length of 8 bytes for each.

T1 traces references to 4-byte fields within an 80-byte area. The trace shows the contents of a 4-byte fixed-point field beginning at each address to which a reference is made. The contents before and after the reference are shown multiplied by the scale factor (2<sup>-2</sup>).

The NAME Operand: The NAME operand specifies a symbol that is printed as the name of a field dumped by a DUMP DATA or DUMP CHANGES statement. Its use is illustrated by the following statements:

```
D1      DUMP  DATA, TABLE(6), DATAM=CL8, NAME=FUNCTION
D2      DUMP  CHANGES, MATRIX, MATRIX+160, NAME=NEWMATRIX
```

D1 dumps a single 8-byte field located at TABLE(6). FUNCTION is printed as the name of the field.

D2 dumps a 160-byte area, which may contain any number of fields. NEWMATRIX is printed as the name of the first field that is dumped.

**SYMBOL TABLES:** Symbol tables are part of the control information that is passed to the TESTRAN editor by the TESTRAN interpreter. (See Figure 3.) Produced by the assembler, each symbol table describes fields defined in a named, unnamed, dummy, or blank common control section. The TESTRAN editor uses the symbol tables to:

- Determine field formats when the DATAM operand is omitted.
- Provide field names when both the DATAM and the NAME operands are omitted.

A blank common control section is common to two or more object modules, and is therefore represented by more than one symbol table. To print fields defined in a common control section, the TESTRAN editor identifies the object module in which the test point was located, and uses the symbol table for the control section as defined in that module.

Except in the case of a blank common control section, the symbol tables define only one format for a given area of storage. They do not define the format of fields that are overlapped by other fields, as in the following sequence:

```
LONGFLT DS D
SHORTFLT DS E
          ORG *-8
ADRLONG DC A(LONGFLT)
ADRSHORT DC A(SHORTFLT)
```

This sequence defines fields that together occupy three full words. LONGFLT occupies the first two words, the second of which is overlapped by ADRLONG. SHORTFLT occupies the third word and is overlapped by ADRSHORT. If the three words were dumped, the first would be printed in default format, and the second and third would be printed as normal address constants.

**DEFAULT:** The fields described in the symbol tables are storage areas and constants defined by assembler DS and DC statements. Instructions are described only if named, and are therefore assumed to be the contents of any program area whose format is not defined in the tables. The area contents are analyzed for operation codes, which are used to determine the printing format for each instruction.

Unless treated as a dummy control section, an allocated area of main storage is not represented by a symbol table. By default, data from such an area is printed in 4-byte hexadecimal fields. Data from registers, including floating-point registers, is also printed in this format.

## HOW TO TRACE CONTROL FLOW AND REFERENCES TO DATA

Suppose that the following sequence is the program to be traced:

```
BEGIN    SAVE    (14,12)
        .
        .
REPEAT   ST      6,MYDATA
        .
        .
DECIDE   BC      4,REPEAT
CONTINUE CALL    ROUTINE1
        .
        .
NEXTSTEP SR      5,5
        .
        .
MYDATA   DC      F'0'
        .
        .
        END    BEGIN
```

The problem is to trace control flow from BEGIN to NEXTSTEP and to trace references to the area beginning at MYDATA. The traces are to be started at BEGIN and are to be stopped at NEXTSTEP.

The next sequence shows a solution:

```
NEWBEGIN TEST    OPEN,BEGIN
          TEST    AT,BEGIN
          TRACE   FLOW,BEGIN,NEXTSTEP
          TRACE   CALL,CONTINUE,NEXTSTEP
          TRACE   REFER,MYDATA,MYDATA+72
          TEST    AT,NEXTSTEP
          TRACE   STOP
BEGIN    SAVE    (14,12)
        .
        .
REPEAT   ST      6,MYDATA
        .
        .
DECIDE   BC      4,REPEAT
CONTINUE CALL    ROUTINE1
        .
        .
NEXTSTEP SR      5,5
        .
        .
MYDATA   DC      F'0'
        .
        .
        END    NEWBEGIN
```

Execution begins at NEWBEGIN, where a TEST OPEN statement establishes BEGIN and NEXTSTEP as test points. NEWBEGIN passes control to BEGIN, where three traces are started:

- The TRACE FLOW statement starts a trace of branches and supervisor calls to, from, or within the area from BEGIN to NEXTSTEP.
- The TRACE CALL statement starts a trace of subroutine calls by CALL macro-instructions located between CONTINUE and NEXTSTEP.
- The TRACE REFER statement traces references by instructions that could change data in the 72-byte area beginning at MYDATA.

To perform these traces, the TESTRAN interpreter retains control and executes the program interpretively, starting at BEGIN. At NEXTSTEP, the traces are stopped and execution continues normally.

The printed output of the three traces can be represented, in abbreviated fashion, as follows:

<u>Output</u>	<u>Recorded During Execution of:</u>
AT LOCATION BEGIN...	} TESTRAN Statements
...TRACE FLOW... STARTED	
...TRACE CALL... STARTED	
...TRACE REFER... STARTED	} Problem Program
...TRACE REFER...TO MYDATA...FROM REPEAT... BEFORE +0 AFTER +16	
...TRACE FLOW...FROM DECIDE...TO REPEAT...CC=4	
.	
.	
...TRACE CALL...TO ROUTINE1...AT CONTINUE...	} TESTRAN Statements
.	
.	
AT LOCATION NEXTSTEP...	
...TRACE STOP,ALL	

The output shows that the traces were started at BEGIN and stopped at NEXTSTEP. It shows that the following events occurred during execution of the problem program:

- A reference was made to MYDATA by REPEAT, resulting in a new value of +16.
- A branch was made from DECIDE to REPEAT on condition code 4.
- A call was made to ROUTINE1 from CONTINUE.

Complete output, as actually printed by the TESTRAN editor, would also show the images of certain instructions, the values of symbolic addresses, and the contents of pertinent registers.

Suppose now that only the traces of control flow should be stopped at NEXTSTEP, and that the trace of references should be continued until the end of the program. The TESTRAN statements should then be written as follows:

```
NEWBEGIN TEST OPEN,BEGIN
          TEST AC,BEGIN
TRACE#1  TRACE FLOW,BEGIN,NEXTSTEP
TRACE#2  TRACE CALL,CONTINUE,NEXTSTEP
          TRACE REFER,MYDATA,MYDATA+72
          TEST AC,NEXTSTEP
          TRACE STOP,(TRACE#1,TRACE#2)
```

The TRACE STOP statement here stops only the traces started by the statements TRACE#1 and TRACE#2. The TESTRAN interpreter continues its interpretive execution of the problem program, and records references to the area at MYDATA until termination of the task (job step).

The TRACE STOP statement speeds up execution by reducing the number of traces. While any trace is in effect, the TESTRAN interpreter must examine each instruction before it is executed to determine whether it will cause some event, such as a branch, that must be recorded. This interpretive execution is necessarily slow, and the time it requires is reduced by stopping each trace when it is no longer needed.

Testing efficiency is also increased by limiting the size of storage areas specified in TRACE statements. For example, if there were three adjoining areas, all could be specified as a single area in a single statement; however, if only the first and third areas were of real interest, it would be better to eliminate output from the second area by using two TRACE statements to specify the first and third areas separately.

With respect to limiting traces, the following specific limits should be kept in mind:

- A trace area should lie entirely within a single control section or allocated storage area. If it does not, the area may be distorted by scatter loading of control sections or by variation in the relative locations of separately allocated areas. Also, if a trace area begins in one control section and ends in another, only data from the first control section can be formatted properly.
- No more than ten traces (corresponding to ten TRACE statements) can be performed simultaneously. If an eleventh trace is started, the tenth trace -- the one most recently started -- is stopped automatically.

A stopped trace can be restarted by executing again (at a later test point) the TRACE CALL, TRACE FLOW, or TRACE REFER statement that originally started the trace. In the same way, an active trace can be shifted to a new area if the area is specified by indexed addresses whose values have changed since the trace was started.

Traces of Asynchronous Exit Routines: Traces are stopped automatically when any of the following routines is entered:

- The end of task exit routine specified by the ETXR operand of an ATTACH macro-instruction.
- The timer completion exit routine specified by a STIMER macro-instruction.
- The error analysis exit routine specified by the SYNAD operand of a DCB macro-instruction.

To trace execution of one of these routines, it is necessary to start traces at a test point within the routine. When the routine returns control to the control program, these traces are automatically stopped and the traces stopped on entry to the routine are automatically restarted.

Traces are not stopped on entry to the program interruption exit routine specified by a SPIE macro-instruction.

Use of Dummy Control Sections: The programmer can trace references to fields of dummy control sections by using the general technique described in "How to Dump a Dummy Control Section." If he assigns varying locations to the dummy control section, he can shift the trace from one location to the next as in the following example:

```

NEWSTART TEST OPEN,START
          TEST AT,LOADBASE+2
          USING RECORD,6
          TRACE REFER, ID,DATE+5,DSECT=RECORD
          .
          .
          .
START    SAVE (14,12)
          .
          .
          .
GETNEXT  GET MYDCB
LOADBASE LR 6,1
          USING RECORD,6
          .
          .
          .
          PUTX MYDCB
          B GETNEXT
          .
          .
          .
MYDCB DCB DSORG=PS,MACRF=(GL,PL),DDNAME=MYDD
RECORD DSECT
ID DS XL4
          .
          .
          .
DATE DS PL5
END NEWSTART

```

GETNEXT uses register 1 to point to a buffer that contains a record to be updated. The program assigns the buffer location to RECORD, a dummy control section that describes the record format. After processing the record, the program replaces it in the data set and executes the same set of instructions to update the next record. On each loop, the TRACE REFER statement is executed immediately after LOADBASE makes RECORD addressable. When first executed, it starts a trace of references to the buffer containing the first record; on each subsequent execution, it shifts the trace to the buffer containing the next record.

## HOW TO COMMENT THE TESTRAN LISTING

A TESTRAN listing can become difficult to interpret when it contains many individual dumps and traces. To make the listing easier to interpret, the programmer can introduce comments that explain or call attention to particular items.

The programmer specifies a comment as an operand of a special DUMP statement (DUMP COMMENT) or in a special operand of a TRACE CALL, TRACE FLOW, or TRACE REFER statement. The following example illustrates both methods:

```
.  
. .  
TEST AT,PAYROLL  
TRACE CALL,CALLFICA,NEXTSTEP,COMMENT='TRACE OF CALLS TO PAYROL-  
L SUBROUTINES'  
TEST AT,TESTCODE-4  
DUMP COMMENT,'G''15'' CONTAINS FICA RETURN CODE'  
DUMP PANEL,G'15'  
. .  
.
```

The comment TRACE OF CALLS TO PAYROLL SUBROUTINES is printed with all output produced by the TRACE CALL statement. The comment G'15' CONTAINS FICA RETURN CODE is printed immediately before the dump of register 15.

(Note that the apostrophes in the second comment are each represented by a pair of apostrophes in the statement. This representation is necessary because apostrophes are used to delimit the comment; for other reasons, ampersands must be represented in the same way.)

## HOW TO CLASSIFY TEST INFORMATION FOR SELECTIVE RETRIEVAL

To avoid printing large quantities of test output, the programmer can divide the output into several classes that can be retrieved selectively. By means of a job control statement, he can select one or more classes for printing immediately after execution of his program. From this information he can decide what other classes he needs for his evaluation of the program. He can then select these classes by submitting a new job that reprocesses the TESTRAN data set.

To classify output, the programmer writes a special operand (SELECT) in one of the following statements:

- TEST OPEN
- TEST AT
- Any DUMP or TRACE statement

Depending on where it appears, the SELECT operand classifies:

- Information recorded at the test points established by a TEST OPEN statement.
- Information recorded at the test point(s) specified in a TEST AT statement.
- Information recorded by an individual DUMP or TRACE statement.

The SELECT operand classifies information by means of a class identification number (an integer from 1 to 8), as in the following statement:

```
T1 TEST OPEN,ENTRY,SELECT=8
```

All information recorded at the test points established by this statement belongs to class 8, except for information that is reclassified by a TEST AT, DUMP, or TRACE statement. Thus, if T1 is followed by

```
TEST AT,PROCESS,SELECT=6
```

all information recorded at PROCESS belongs to class 6, except for information that is reclassified by a DUMP or TRACE statement, such as:

```
DUMP DATA,MYDATA,SELECT=5
```

The dump of MYDATA belongs to class 5, and only to class 5. As a result of reclassification, it does not belong to either class 6 or class 8.

Use of the SELECT operand does not imply that all information must be classified. Unclassified as well as classified information can be selected for printing.

### TESTING OF COMPLEX PROGRAMS

This part of Section 2 describes the testing of programs that are not simply structured or are not formed from single object modules.

#### HOW TO TEST A MODULE ALREADY IN A LIBRARY

As stated in Section 1, TESTRAN statements and the problem program can be assembled together or separately. Assembling the two together is usually the more convenient, but the sophisticated programmer may discover cases where separate assembly is more efficient. For example, the programmer may have assembled and tried to execute a program before deciding to use TESTRAN. If he has saved the program in a library, he may wish to assemble TESTRAN statements separately to avoid reassembling the program to be tested.

Separate assembly presents two major problems. First, there is no simple symbolic way that TESTRAN statements can refer to locations in the problem program. Second, assuming that the object or load module in the library contains no symbol tables, there is no simple way of obtaining TESTRAN output in the proper symbolic format.

References to the Problem Program: There are three ways that TESTRAN statements can refer to locations in the problem program. The first, which is the only way that can be used in TEST OPEN and TEST AT statements, is to write each address as an external reference plus or minus an appropriate displacement. The external reference is a symbol defined in the problem program and listed in the external symbol dictionary (the first part of the assembly listing). The displacement is the number of bytes from the location named by the symbol to the location of the operand; it can be calculated from the object code addresses contained in the assembly listing.

The second way of referring to the problem program is by explicit addresses. These can be written to use base registers loaded by the problem program. Displacements from base addresses can be calculated from the object code addresses in the assembly listing.

The third way of referring to the problem program is to use dummy control sections that describe the format of the problem program. The name of each must be declared as the address in a base register that is loaded by the problem program. Areas defined in the dummy control sections (which correspond to areas in the problem program) can then be referred to symbolically by DUMP DATA, DUMP CHANGES, and TRACE REFER statements that are written with DSECT operands.

Output Format: The output of DUMP DATA, DUMP CHANGES, and TRACE REFER statements is printed as four-byte hexadecimal fields unless each statement contains a DATAM or DSECT operand. The DATAM operand specifies a uniform field format for all data in the area specified by the statement. The DSECT operand specifies use of the symbol table for a dummy control section that is assembled with the TESTRAN statements.

Symbol tables are optional features of assembly and linkage editing, and are requested by means of job control statements. If the programmer anticipated the use of TESTRAN, he could have requested symbol tables when the problem program module was created. The tables for the problem program could then be used to determine the output format.

Linkage Editing and Execution: After being assembled, the TESTRAN module (TESTRAN statements and dummy control sections) is processed by the linkage editor. The programmer must provide linkage editor control statements to combine this module with the problem program module. For example, the statements:

```
INCLUDE MYLIB(MYPROG)
ENTRY   NEWSTART
NAME    MYPROG(R)
```

specify that the load module is to include the load module MYPROG from the library MYLIB; that the entry point is to be NEWSTART (assumed to be the name of a TEST OPEN statement); and that the new load module is to replace the original problem program module in the library.

The normal procedure is followed in executing the new load module and printing the TESTRAN output. If the output shows an error in a particular control section, the programmer can replace the control section with a new one through use of the linkage editor. Since a symbol table can be requested when assembling the new control section, the programmer may wish to eliminate DATAM or DSECT operands in TESTRAN statements that refer to the control section. If so, he assembles a complete new set of TESTRAN statements, which form an implicit control section named after the TEST OPEN statement. If each new control section is named after the control section it replaces, the replacement is automatic, and only two linkage editor control statements are needed:

```
INCLUDE MYLIB(MYPROG)
NAME    MYPROG(R)
```

When the new load module is tested, the TESTRAN output may show an error in one of the replacement control sections. If there is a symbol table for this control section, the control section should not be replaced with another of the same name. The linkage editor does not replace symbol tables when it replaces control sections; therefore, the table originally associated with each section name remains in effect.

Test Completion: When testing is completed, the programmer can direct the linkage editor to prepare the load module for productive use. For example, he might write the following control statements:

```
ENTRY   START
REPLACE NEWSTART
INCLUDE MYLIB(MYPROG)
NAME    MYPROG(R)
```

These statements restore the normal entry point (START) and delete the TESTRAN control section (NEWSTART). Symbol tables in the module are deleted as a result of omitting an option in a job control statement.

## HOW TO ENLARGE ON A PARTIALLY TESTED PROGRAM

Suppose that the following program has been tested successfully:

```
TESTMOD1 TEST  OPEN,MOD1
      .
      .
MOD1    CSECT
      .
      .
      .
      END  TESTMOD1
```

MOD1 is now to become a subroutine of another control section, MOD2, and the two control sections are to be tested together. The enlarged program is as follows:

```
TESTMOD2 TEST  OPEN,MOD2,OPTTEST=TESTMOD1
      .
      .
MOD2    CSECT
      .
      .
CALLMOD1 CALL  MOD1
      .
      .
      .
TESTMOD1 TEST  OPEN,MOD1
      .
      .
MOD1    CSECT
      .
      .
      .
      END  TESTMOD2
```

Execution begins at TESTMOD2, the first of a group of TESTSTRAN statements for testing MOD2. In effect, this statement executes the statement TESTMOD1; as a result, it establishes test points as specified by TEST AT statements following both TESTMOD1 and TESTMOD2. TESTMOD2 ignores the second operand (MOD1) of TESTMOD1 and passes control to the problem program at MOD2.

Because MOD1 has been tested previously, test information about MOD1 is simply insurance against unexpected errors. The programmer may therefore wish to defer printing this information until after he has examined the information about MOD2. If so, he can classify the information about MOD2 and select only this information for immediate printing. He can save the data set that contains the information and, if it proves necessary, select the information about MOD1 at a later date.

The programmer classifies information about MOD2 by means of a special operand (SELECT) described in "How to Classify Test Information for Selective Retrieval." There are several statements in which he can write this operand, but for the present purpose he can best write it in the TEST AT statements that follow TESTMOD2:

```

TESTMOD2 TEST OPEN,MOD2,OPTEST=TESTMOD1
TEST AT,MOD2,SELECT=8
.
.
.
TEST AT,CALLMOD1,SELECT=8
.
.
.

```

The programmer can select information about MOD2 by specifying class 8 in a job control statement, as explained in Section 3. In a later job, he can repeat the editing of TESTSTRAN output and select unclassified output to print information about MOD1.

The SELECT operands in the TEST AT statements classify information recorded at test points in MOD2. A SELECT operand in the statement TESTMOD2 would provide the same function if that statement did not include the operand OPTEST=TESTMOD1. In a TEST OPEN statement, a SELECT operand classifies information recorded at all test points established by the statement, including those established as the result of an OPTEST operand. A SELECT operand in TESTMOD2 would therefore classify information recorded at test points in both MOD2 and MOD1. It would do so even if a different SELECT operand (e.g., SELECT=7) were written in TESTMOD1, because the operands of a TEST OPEN statement are ignored if the statement is not actually executed.

#### HOW TO TEST AN OVERLAY PROGRAM

An overlay program is a load module that is divided into several overlay segments. For testing purposes, each segment must be treated as a separate program. That is, it must contain its own TESTSTRAN statements, beginning with a TEST OPEN statement. During execution, only one TEST OPEN statement can receive control; it must be located in the root segment, and it must contain a special operand (OPTEST) that points to all other TEST OPEN statements, as in the following example:

```

Segment 1 { TESTSEG1 TEST OPEN,ENTRY,OPTEST=(TESTSEG2,TESTSEG3)
(Root Segment) { .
.
.
Segment 2 { TESTSEG2 TEST OPEN
.
.
.
Segment 3 { TESTSEG3 TEST OPEN
.
.
.
END TESTSEG1

```

Except for references by the OPTEST operand, symbolic references between segments are not allowed in TESTSTRAN statements. External references must be declared in assembler EXTRN and ENTRY statements.

A TEST OPEN statement and the TESTSTRAN statements that follow it form an implicit TESTSTRAN control section that must be inserted in the proper overlay segment. Thus, for the example just given, the programmer might write the following linkage editor control statements:

```

INSERT TESTSEG1,...
OVERLAY ROOTNODE
INSERT TESTSEG2,...
OVERLAY ROOTNODE
INSERT TESTSEG3,...

```

When a segment is overlaid, traces started by TRACE statements in the segment are automatically stopped. They are not automatically restarted when the segment is reloaded, but are restarted when the TRACE statements are again executed at a test point in the segment. To ensure that traces are restarted, the programmer must therefore design his testing logic so that TRACE statements are executed each time a segment is entered after being overlaid and reloaded.

#### HOW TO TEST A DYNAMIC SERIAL PROGRAM

A dynamic serial program is a combination of two or more load modules that are loaded and executed as a single task. Each load module can contain TESTSTRAN statements; if it does, however, it is neither reenterable nor serially reusable.

A module that is not reusable is normally loaded each time it is entered by a supervisor assisted linkage. For this reason, a TEST OPEN statement must be executed to establish test points each time the module is entered by means of a LINK, XCTL, or ATTACH macro-instruction. Before control is passed or returned to another module, testing of the module should be stopped by a TEST CLOSE statement, as in the following example:

```

NEWENTRY TEST OPEN,ENTRY
      *
      *
      *
      TEST AT,FINISH
      TEST CLOSE
ENTRY  SAVE (14,12)
      *
      *
      *
FINISH RETURN (14,15)
      END NEWENTRY

```

At the test point FINISH, the TEST CLOSE statement nullifies the effect of the TEST OPEN statement and returns control to the test point. As a result, the TESTSTRAN interpreter releases storage areas acquired for internal functions. If not released, the areas would be duplicated the next time the module was loaded and tested.

A module is not loaded each time it is entered if it is already in storage and either of these conditions is met:

- The program was loaded by a LOAD macro-instruction and is not currently being used in a supervisor assisted linkage.
- The program is entered by means of an ATTACH macro-instruction at an entry point identified by an IDENTIFY macro-instruction.

If the module is loaded only once, the TEST CLOSE statement need not be used, and a TEST OPEN statement need be executed only once.

When a supervisor assisted linkage is made to another module, all traces are automatically stopped. They are not automatically restarted when control is returned, but can be restarted by appropriate TRACE statements. The TRACE statements should follow a TEST AT statement that specifies the return address as a test point.

### SECTION 3: HOW TO WRITE JOB CONTROL STATEMENTS

To use TESTRAN, the programmer must write job control statements to define the job to be performed by the operating system. A typical TESTRAN job consists of one or more job steps, each of which performs one of the following functions:

- Assembly of the problem program
- Linkage editing of the problem program
- Execution of the problem program
- Editing of test information

The job control statements used in defining jobs and job steps are described in the publication IBM System/360 Operating System: Job Control Language. Statements for performing specific TESTRAN-oriented jobs are listed below. The jobs defined by these model job definitions include the following job steps:

- Assembly
- Linkage Editing
- Execution
- TESTRAN Editing
- Assembly and Linkage Editing
- Assembly, Linkage Editing, and Execution
- Assembly, Linkage Editing, Execution, and TESTRAN Editing

Most of the model job definitions refer to IBM-supplied cataloged procedures, which are defined in Appendix B. Before attempting to use these procedures, the programmer should make certain that they have been included in the procedure library at his installation. If a procedure has been omitted, the programmer can copy the necessary statements from the appendix.

#### ASSEMBLY

Figure 5 defines a job that executes the F-level assembler program. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below. The shaded statement is optional.

```
1. //jobname      JOB  job parameters
2. //            EXEC PROC=ASMEC, PARM=TEST
3. //ASM.SYSPUNCH DD  DSNAME=dsname, UNIT=SYSDA, SPACE=(80,(200,50)),DISP=(NEW,CATLG)
4. //ASM.SYSIN    DD  data definition parameters
```

Figure 5. Job Control Statements for Assembly

1. This statement provides general job control information.
2. This statement refers to a cataloged procedure named ASMEC, which defines a single job step named ASM. The PARM parameter specifies the option TEST, which causes symbol tables to be included in the object module. The PARM parameter implies the following options:

```
DECK
LIST
XREF
LINECNT=standard line count
```

If desired, other options can be specified in place of the implied options. The TEST option, however, must be specified.

3. This statement is optional. If present, it saves the object module as a cataloged data set in direct-access storage. The data set can subsequently be referred to by name as primary or additional input to the linkage editor.

Statement 3 overrides the following statement in the procedure ASMEC:

```
//SYSPUNCH DD UNIT=SYSCP
```

If this statement is not overridden, it causes the object module to be produced as a deck of punched cards.

4. This statement defines a data set that contains the source program to be assembled. This data set can appear in the input stream.

### LINKAGE EDITING

Figure 6 defines a job that executes the largest linkage editor program available at the installation. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below. The shaded statement is optional.

```
1. //jobname JOB job parameters
2. // EXEC PROC=LKED, PARM=TEST
3. //LKED.SYSLMOD DD DSNNAME=libname(member), DISP=(MOD,CATLG)
4. //LKED.SYSIN DD data definition parameters
5. //LKED.ddname DD data definition parameters
```

Figure 6. Job Control Statements for Linkage Editing

1. This statement provides general job control information.
2. This statement refers to a cataloged procedure named LKED, which defines a single job step that is also named LKED. The PARM parameter specifies the option TEST, which causes symbol tables and object module control dictionaries to be included in the load module. Additional options that can be specified are:

```
SCTR or OVLY
DC
LIST
XREF or MAP
NCAL
LET or XCAL
```

Of these, LIST and XREF (which includes MAP) are diagnostic options, and NCAL and LET (which includes XCAL) are special processing options that are useful in testing a program.

Because the TEST option must be specified, the NE and REUS or RENT options cannot be specified. The load module is therefore editable and not reusable.

3. This statement is optional. If present, it saves the load module as a member of a new cataloged partitioned data set (library). The

data set may be new or may already exist; if it exists, the load module replaces any other member of the data set that has the same member name. If the data set has already been cataloged, the DISP parameter should be omitted.

The load module can be referred to by its member name for subsequent execution as a program or for reprocessing by the linkage editor. The saved load module should not be reprocessed, however, if the reprocessing involves replacing any non-TESTSTRAN control section with another control section of the same name. Such a control section would continue to be represented by the symbol table and control dictionaries for the object module to which it originally belonged. Data recorded from this control section would therefore not be printed in the proper symbolic format.

Statement 3 overrides the DSNNAME and DISP parameters of the following statement in the procedure LKED:

```
//SYSLMOD DD DSNNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),
//          UNIT=SYSDA,DISP=(MOD,PASS)
```

If these parameters are not overridden, they cause the load module to be produced as a member of a temporary data set that is deleted at the end of the job.

4. } These statements define the input to the linkage editor. Statement  
 5. } 4 defines the primary input, which is a data set containing one or more object modules, or linkage editor control statements, or both.

Statement 5 is optional. If present, it defines either an included data set or an automatic call library. It can be repeated as necessary to define any number of input data sets.

Sequentially organized data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step LKED.

## EXECUTION

Figure 7 defines a job that executes a program for testing by the TESTSTRAN interpreter. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below. The shaded statements are optional.

1.	//jobname	JOB	job parameters
2.	//JOBLIB	DD	DSNAME=libname,DISP=(OLD,PASS),UNIT=SYSDA,VOLUME=SER=vserial
3.	//	EXEC	PGM=member
4.	//SYSTEST	DD	DSNAME=dsname,UNIT=SYSSQ,SPACE=(300,(100)),DISP=(NEW,CATLG)
5.	//SYGABEND	DD	UNIT=SYSDA,SYSOBT=A
6.	//ddname	DD	data definition parameters

Figure 7. Job Control Statements for Execution

1. This statement provides general job control information.
2. This statement is optional. If present, it points to a private load module library that is to be used as the job library. If this library has been cataloged, the UNIT and VOLUME parameters should be omitted.

3. This statement refers to a load module that is a member of either the system link library or the job library.
4. This statement saves the output of the TESTRAN interpreter as a cataloged data set. This data set can subsequently be referred to by name for processing by the TESTRAN editor.
5. This statement is optional. If present, it defines a data set to contain an abnormal termination dump.
6. This statement is optional. If present, it defines a data set that is used by the problem program. It can be repeated as necessary to define any number of data sets.

Sequentially organized input data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the job step.

### TESTRAN EDITING

Figure 8 defines a job that executes the TESTRAN editor. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below.

```

1. //jobname      JOB  job parameters
2. //            EXEC PROC=TTED
3. //EDIT.SYSTEST DD  DSN=dsname,UNIT=SYSSQ,SEP=SYSUT1,DISP=(OLD)

```

Figure 8. Job Control Statements for TESTRAN Editing

1. This statement provides general job control information.
2. This statement refers to a cataloged procedure named TTED, which defines a single job step named EDIT.

TESTRAN Editor Options: Three options can be specified by a PARM parameter written as:

PARM=[\*][Ta]...[Pb]

\*  
increases the speed of TESTRAN editing by a factor of four. At the same time, it increases main storage requirements from 18K bytes to 50K bytes. If present, it must occupy the first position in the parameter.

Ta  
identifies a class of test information that is to be edited. The value a is either an unsigned decimal integer from 1 to 8, a blank, or the letter A. If an integer, it is a class identification number specified by a SELECT keyword operand in one or more TEST OPEN, TEST AT, DUMP or TRACE statements. If a blank, it indicates that all unclassified data is to be edited. If the letter A, it indicates that all data is to be edited, regardless of classification.

The subfield Ta can be repeated as many times as necessary to select all desired information for processing during a single execution of the TESTRAN editor. Note that if a class of

information is not selected, and has not previously been edited, the input TESTRAN data set should be saved to allow later editing of this information.

If the subfield Ta is omitted, all information is printed as if TA were specified

Pb

specifies the maximum number of pages to be printed. The value b is an unsigned decimal integer. It must not be greater than the maximum page count established at the installation during system generation.

If the subfield Pb is omitted, the maximum count is as specified in the first TEST OPEN statement executed under the task (job step) that created the data set. If this TEST OPEN statement did not specify a maximum, the installation maximum is assumed.

3. This statement defines the input TESTRAN data set, which contains the test information to be edited. If all of the information is to be edited (rather than just selected classes), the disposition should be changed to DISP=(OLD,DELETE).

### ASSEMBLY AND LINKAGE EDITING

Figure 9 defines a job that executes the E-level assembler program and the largest linkage editor program available at the installation. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below. The shaded statements are optional.

```
1. //jobname      JOB  job parameters
2. //            EXEC PROC=TASME
3. //ASM.SYSPUNCH DD  DSNAME=dsname,DISP=(MOD,PASS)
4. //ASM.SYSIN    DD  data definition parameters
5. //LKED.SYSLIN  DD  DSNAME=dsname,DISP=(OLD,CATLG)
6. //LKED.SYSLMOD DD  DSNAME=libname(member),DISP=(MOD,CATLG)
7. //LKED.SYSIN   DD  data definition parameters
8. //LKED.dname  DD  data definition parameters
```

Figure 9. Job Control Statements for Assembly and Linkage Editing

1. This statement provides general job control information.
2. This statement refers to a cataloged procedure named TASME, which defines two job steps: ASM and LKED.

Assembler Options: The following assembler options are specified or implied in the cataloged procedure:

```
TEST
LOAD
LIST
XREF
LINECNT=standard line count
```

The TEST option is required to cause symbol tables to be included in the object module. The LOAD option indicates that the object module is to be stored on an external storage device. The last three options are standard default options; of these, LIST and XREF are diagnostic options useful in program testing.

The default options can be overridden by writing:

```
PARM.ASM=(TEST,LOAD,overriding options)
```

where the overriding options are any of the following:

```
NOLIST  
NOXREF  
LINECNT=nn
```

where nn is an unsigned decimal integer from 1 to 99. Any default option not overridden remains in effect. The TEST and LOAD options, because they are not default options, must be explicitly specified.

Linkage Editor Options: The following linkage editor options are specified in the cataloged procedure:

```
TEST  
LIST  
XREF  
NCAL  
LET
```

The TEST option is required to cause the symbol tables and object module control dictionaries to be included in the load module. LIST and XREF are diagnostic options, and NCAL and LET are special processing options that are useful in program testing.

These options can be respecified by writing:

```
PARM.LKED=(TEST,respecified options)
```

where the respecified options are any of the following:

```
SCTR or OVLY  
DC  
LIST  
XREF or MAP  
NCAL  
LET or XCAL
```

Each of the original options (TEST, LIST, XREF, NCAL, and LET) is overridden if it is not respecified. Because TEST must be respecified, the REUS, RENT, and NE options cannot be specified.

3. This statement is optional, but, if it is written, statement 5 must also be written. The two statements together save the object module produced by the assembler as a cataloged data set in direct-access storage. This data set can later be referred to by name as primary or additional input to the linkage editor.

Statements 3 and 5 override the DSNAME and DISP parameters of the following statements in the procedure TASME:

```
//SYSPUNCH DD DSNAME=&LOADSET,UNIT=SYSDA,  
//          SPACE=(80,(200,50)),DISP=(MOD,PASS)  
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD)
```

If these parameters are not overridden, they cause the object module to be produced as a temporary data set that is deleted at the end of the job.

If statements 3 and 5 are present, statement 4 must appear between them in the sequence.

4. This statement defines the data set that contains the source program to be assembled. This data set can appear in the input stream.
5. Refer to paragraph 3 above.
6. This statement is optional. If present, it saves the load module as a member of a cataloged partitioned data set (library). The data set may be new or may already exist; if it exists, the load module replaces any other member of the data set that has the same member name. If the data set has already been cataloged, the DISP parameter should be omitted.

The load module can be referred to by its member name for later execution as a program or for reprocessing by the linkage editor. The saved load module should not be reprocessed, however, if the reprocessing involves replacing any non-TESTRAN control section with another control section of the same name. Such a control section would continue to be represented by the symbol table and control dictionaries for the object module to which it originally belonged. Data recorded from this control section would therefore not be printed in the proper symbolic format.

Statement 6 overrides the DSNAME and DISP parameters of the following statement in the procedure TASME:

```
//SYSLMOD DD DSNAME=&GOSSET(GO),SPACE=(1024,(50,20,1)),  
// UNIT=SYSDA,DISP=(MOD,PASS)
```

If these parameters are not overridden, they cause the load module to be produced as a member of a temporary data set that is deleted at the end of the job.

7. } These statements are optional. If present, they define input to
8. } the linkage editor.

Statement 7 defines a data set to be concatenated with the primary input to the linkage editor. (The primary input is the object module produced by the assembler; refer to paragraph 5 above.)

Statement 8 defines either an included data set or an automatic call library. It can be repeated as necessary to define any number of input data sets.

Sequentially organized data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step LKED.

#### ASSEMBLY, LINKAGE EDITING, AND EXECUTION

Figure 10 defines a job that executes the E-level assembler program, the largest linkage editor program available at the installation, and the load module produced by the linkage editor. The statements in the figure are numbered, and are explained in correspondingly numbered paragraphs below. The shaded statements are optional.

1.	//jobname	JOB	job parameters
2.	//JOBLIB	DD	DSNAME=libname, DISP=(OLD, PASS), UNIT=SYSDA, VOLUME=SER=vserial
3.	//	EXEC	PROC=TASMEG
4.	//ASM.SYSPUNCH	DD	DSNAME=dsname, DISP=(MOD, PASS)
5.	//ASM.SYSIN	DD	data definition parameters
6.	//LKED.SYSLIN	DD	DSNAME=dsname, DISP=(OLD, CATLG)
7.	//LKED.SYSLMOD	DD	DSNAME=libname(member), DISP=(MOD, CATLG)
8.	//LKED.SYSIN	DD	data definition parameters
9.	//LKED.ddname	DD	data definition parameters
10.	//GO.SYSTEST	DD	DSNAME=dsname, UNIT=SYSSQ, SPACE=(300, (100)), DISP=(NEW, CATLG)
11.	//GO.SYSABEND	DD	UNIT=SYSDA, SYSOUT=A
12.	//GO.ddname	DD	data definition parameters

Figure 10. Job Control Statements for Assembly, Linkage Editing, and Execution

1. This statement provides general job control information.
2. This statement is optional. If present, it points to a private load module library that is to be used as the job library. If this library has been cataloged, the UNIT and VOLUME parameters should be omitted.
3. This statement refers to a cataloged procedure TASMEG that defines three job steps: ASM, LKED, and GO.

Assembler Options: The following assembler options are specified or implied in the cataloged procedure:

```
TEST
LOAD
LIST
XREF
LINECNT=standard line count
```

The TEST option is required to cause symbol tables to be included in the object module. The LOAD option indicates that the object module is to be stored on an external storage device. The last three options are standard default options; of these, LIST and XREF are diagnostic options useful in program testing.

The default options can be overridden by writing:

```
PARM.ASM=(TEST,LOAD,overriding options)
```

where the overriding options are any of the following:

```
NOLIST
NOXREF
LINECNT=nn
```

where nn is an unsigned decimal integer from 1 to 99. Any default option not overridden remains in effect. The TEST and LOAD options, because they are not default options, must be explicitly specified.

Linkage Editor Options: The following linkage editor options are specified in the cataloged procedure:

```
TEST
LIST
XREF
NCAL
LET
```

The TEST option is required to cause the symbol tables and object module control dictionaries to be included in the load module. LIST and XREF are diagnostic options, and NCAL and LET are special processing options that are useful in program testing.

These options can be specified by writing:

```
PARM.LKLD=(TEST, respecified options)
```

where the respecified options are any of the following:

```
SCTR or OVLY  
DC  
LIST  
XREF or MAP  
NCAL  
LET or XCAL
```

Any of the original options (TEST, LIST, XREF, NCAL, and LET) that is not respecified is overridden. Because the TEST option must be respecified, the REUS, RENT, and NE options cannot be specified.

Problem Program Information: Information can be passed to the problem program by writing:

```
PARM.GO=(xxx...)
```

where xxx... is the information.

4. This statement is optional, but, if it is written, statement 6 must also be written. The two statements together save the object module produced by the assembler as a cataloged data set in direct-access storage. This data set can later be referred to by name as primary or additional input to the linkage editor.

Statements 4 and 6 override the DSNAME and DISP parameters of the following statements in the procedure TASMEG:

```
//SYSPUNCH DD DSNAME=&LOADSET,UNIT=SYSDA,  
//          SPACE=(80,(200,50)),DISP=(MOD,PASS)  
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD)
```

If these parameters are not overridden, they cause the object module to be produced as a temporary data set that is deleted at the end of the job.

If statements 4 and 6 are present, statement 5 must appear between them in the sequence.

5. This statement defines a data set that contains the source program to be assembled. This data set can appear in the input stream.
6. Refer to paragraph 4 above.
7. This statement is optional. If present, it saves the load module as a member of a cataloged partitioned data set (library). The data set may be new or may already exist; if it exists, the load module replaces any other member of the data set that has the same member name. If the data set has already been cataloged, the DISP parameter should be omitted.

The load module can be referred to by its member name for later execution as a program or for reprocessing by the linkage editor. The saved load module should not be reprocessed, however, if the reprocessing involves replacing any non-TESTTRAN control section with

another control section of the same name. Such a control section would continue to be represented by the symbol table and control dictionaries for the object module to which it originally belonged. Data recorded from this control section would therefore not be printed in the proper symbolic format.

Statement 7 overrides the DSNNAME and DISP parameters of the following statement in the procedure TASMEG:

```
//SYSLMOD DD DSNNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),  
//          UNIT=SYSDA,DISP=(MOD,PASS)
```

If these parameters are not overridden, they cause the load module to be produced as a member of a temporary data set that is deleted at the end of the job.

8. } These statements are optional. If present, they define input to  
9. } the linkage editor.

Statement 8 defines a data set to be concatenated with the primary input to the linkage editor. (The primary input is the object module produced by the assembler; refer to paragraph 6 above.)

Statement 9 defines either an included data set or an automatic call library. It can be repeated as necessary to define any number of input data sets.

Sequentially organized data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step LKED.

10. This statement saves the output of the TESTRAN interpreter as a cataloged data set. This data set can later be referred to by name for processing by the TESTRAN editor.
11. This statement is optional. If present, it defines a data set to contain an abnormal termination dump.
12. This statement is optional. If present, it defines a data set that is used by the problem program. It can be repeated as necessary to define any number of data sets.

Sequentially organized input data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step GO.

#### ASSEMBLY, LINKAGE EDITING, EXECUTION, AND TESTRAN EDITING

Figure 11 defines a job that executes the E-level assembler program, the largest linkage editor available at the installation, the load module produced by the linkage editor, and the TESTRAN editor. The statements in the figure are numbered, and are explained in the correspondingly numbered paragraphs below. The shaded statements are optional.

```

1. //jobname      JOB   job parameters
2. //JOBLIB      DD   DSNNAME=libname,DISP=(OLD,PASS),UNIT=SYSDA,VOLUME=SER=vserial
3. //           EXEC  PROC=TASMEGED
4. //ASM.SYSPUNCH DD   DSNNAME=dsname,DISP=(MOD,PASS)
5. //ASM.SYSIN   DD   data definition parameters
6. //LKED.SYSLIN DD   DSNNAME=dsname,DISP=(MOD,CATLG)
7. //LKED.SYSLMOD DD  DSNNAME=libname(member),DISP=(MOD,CATLG)
8. //LKED.SYSIN  DD   data definition parameters
9. //LKED.dsname DD   data definition parameters
10. //GO.SYSTEST  DD   DSNNAME=dsname,DISP=(NEW,CATLG)
11. //GO.SYSABEND DD  UNIT=SYSDA,SYSOBT=A
12. //GO.dsname  DD   data definition parameters
13. //EDIT.SYSTEST DD  DSNNAME=dsname,DISP=(OLD,disp)

```

Note: This sequence produces a load module and executes it as a job step. If the job step terminates normally, the output of the TESTSTRAN interpreter is processed by the TESTSTRAN editor; if the job step terminates abnormally, no editing is performed.

To ensure execution of the TESTSTRAN editor, the job can be divided into two jobs by using the sequences described in "Assembly, Linkage Editing, and Execution" and "TESTSTRAN Editing." The output of the TESTSTRAN interpreter is then edited even if the job step that produces the output terminates abnormally.

Figure 11. Job Control Statements for Assembly, Linkage Editing, Execution, and TESTSTRAN Editing

1. This statement provides general job control information.
2. This statement is optional. If present, it points to a private load module library that is to be used as the job library. If this library has been cataloged, the UNIT and VOLUME parameters should be omitted.
3. This statement refers to a cataloged procedure TASMEGED that defines four job steps: ASM, LKED, GO, and EDIT.

Assembler Options: The following assembler options are specified or implied in the cataloged procedure:

```

TEST
LOAD
LIST
XREF
LINECNT=standard line count

```

The TEST option is required to cause symbol tables to be included in the object module. The LOAD option indicates that the object module is to be stored on an external storage device. The last three options are standard default options; of these, LIST and XREF are diagnostic options useful in program testing.

The default options can be overridden by writing:

```
PARM.ASM=(TEST,LOAD,overriding options)
```

where the overriding options are any of the following:

```

NOLIST
NOXREF
LINECNT=nn

```

where nn is an unsigned decimal integer from 1 to 99. Any default option not overridden remains in effect. The TEST and LOAD options, because they are not default options, must be explicitly specified.

Linkage Editor Options: The following linkage editor options are specified in the cataloged procedure:

TEST  
LIST  
XREF  
NCAL  
LET

Of these, TEST is required to cause the symbol tables and object module control dictionaries to be included in the load module. LIST and XREF are diagnostic options, and NCAL and LET are special processing options that are useful in program testing.

These options can be respecified by writing:

PARM.LKED=(TEST,respecified options)

where the respecified options are any of the following:

SCTR or OVLY  
DC  
LIST  
XREF or MAP  
NCAL  
LET or XCAL

Any of the original options (TEST, LIST, XREF, NCAL, and LET) that is not respecified is overridden. Because the TEST option must be respecified, the REUS, RENT, and NE options cannot be specified.

Problem Program Information: Information can be passed to the problem program by writing:

PARM.GO=(xxx...)

where xxx... is the information.

TESTRAN Editor Options: Two options can be specified by a PARM parameter written as:

PARM.EDIT=[\*][Ta]...[Pb]

\*

increases the speed of TESTRAN editing by a factor of four. At the same time, it increases main storage requirements from 18K bytes to 50K bytes. If present, it must occupy the first position in the parameter.

Ta

identifies a class of test information that is to be edited. The value a is either an unsigned decimal integer from 1 to 8, a blank, or the letter A. If an integer, it is a class identification number specified by a SELECT keyword operand in one or more TEST OPEN, TEST AT, DUMP or TRACE statements. If a blank, it indicates that all unclassified data is to be edited. If the letter A, it indicates that all data is to be edited, regardless of classification.

The subfield Ta can be repeated as many times as necessary to select all desired information for processing during a single execution of the TESTRAN editor. Note that if a class of information is not selected, and has not previously been edited, the input TESTRAN data set should be saved to allow later editing of this information.

If the subfield Ta is omitted, all information is printed as if TA were specified.

Pb

specifies the maximum number of pages to be printed. The value b is an unsigned decimal integer. It must not be greater than the maximum page count established at the installation during system generation.

If the subfield Pb is omitted, the maximum count is as specified in the first TEST OPEN statement executed under the task (job step) that created the data set. If this TEST OPEN statement did not specify a maximum, the installation maximum is assumed.

4. This statement is optional, but, if it is written, statement 6 must also be written. The two statements together save the object module produced by the assembler as a cataloged data set in direct-access storage. This data set can later be referred to by name as primary or additional input to the linkage editor.

Statements 4 and 6 override the DSNAME and DISP parameters of the following statements in the procedure TASMEGED:

```
//SYSPUNCH DD DSNAME=&LOADSET,UNIT=SYSDA,  
//          SPACE=(80,(200,50)),DISP=(MOD,PASS)  
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD)
```

If these parameters are not overridden, they cause the object module to be produced as a temporary data set that is deleted at the end of the job.

If statements 4 and 6 are present, statement 5 must appear between them in the sequence.

5. This statement defines a data set that contains the source program to be assembled. This data set can appear in the input stream.
6. Refer to paragraph 4 above.
7. This statement is optional. If present, it saves the load module as a member of a cataloged partitioned data set (library). The data set may be new or may already exist; if it exists, the load module replaces any other member of the data set that has the same member name. If the data set has already been cataloged, the DISP parameter should be omitted.

The load module can be referred to by its member name for later execution as a program or for reprocessing by the linkage editor. The saved load module should not be reprocessed, however, if the reprocessing involves replacing any non-TESTSTRAN control section with another control section of the same name. Such a control section would continue to be represented by the symbol table and control dictionaries for the object module to which it originally belonged. Data recorded from this control section would therefore not be printed in the proper symbolic format.

Statement 7 overrides the DSNAME and DISP parameters of the following statement in the procedure TASMEGED:

```
//SYSLMOD DD DSNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),  
//          UNIT=SYSDA,DISP=(MOD,PASS)
```

If these parameters are not overridden, they cause the load module to be produced as a member of a temporary data set that is deleted at the end of the job.

8. } These statements are optional. If present, they define input to  
9. } the linkage editor.

Statement 8 defines a data set to be concatenated with the primary input to the linkage editor. (The primary input is the object module produced by the assembler; refer to paragraph 6 above.)

Statement 9 defines either an included data set or an automatic call library. It can be repeated as necessary to define any number of input data sets.

Sequentially organized data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step LKED.

10. This statement is optional. If present, it saves the output of the TESTSTRAN interpreter as a cataloged data set. This data set can be referred to by name for later processing by the TESTSTRAN editor.

Statement 10 overrides the DSNNAME and DISP parameters of the following statement in the procedure TASMEGED:

```
//SYSTEST DD DSNNAME=&TESTSET,SPACE=(300,(100)),  
//          UNIT=SYSSQ,DISP=(NEW,PASS)
```

If these parameters are not overridden, they define a temporary data set that is deleted at the end of the job.

11. This statement is optional. If present, it defines a data set to contain an abnormal termination dump.

12. This statement is optional. If present, it defines a data set that is used by the problem program. It can be repeated as necessary to define any number of data sets.

Sequentially organized input data sets can appear in the input stream. However, in a system with a primary control program or with MFT, only one data set can appear in the input stream, and it must be defined by the last DD statement for the step GO.

13. This statement is optional. If present, it points to a data set defined by statement 10. The data set contains the test information to be edited under the procedure TASMEGED.

If all information is to be edited (rather than just selected classes), the disposition should be changed to DISP=(OLD,DELETE).

Statement 13 overrides the DSNNAME and DISP parameters of the following statement in the procedure TASMEGED:

```
//SYSTEST DD DSNNAME=&TESTSET,UNIT=(SYSSQ,SEP=(SYSUT1)),  
//          DISP=(OLD,DELETE)
```

If these parameters are not overridden, they refer to a temporary data set defined by the previous step of the procedure. This data set is deleted at the end of the job.

## SECTION 4: HOW TO INTERPRET SYSTEM OUTPUT

Every TESTRAN job produces system output that includes listings of job control statements and of certain data sets. The control statements include both those in the input stream and those in cataloged procedures that are invoked in the input stream. The data sets are those to which the job control statements assign a SYSOUT disposition.

Typical system output data sets are abnormal termination dumps and the listings produced by the assembler, the linkage editor, and the TESTRAN editor. This section describes only the last listing; the others are described in the publications:

IBM System/360 Operating System: Messages, Completion Codes and Storage Dumps

IBM System/360 Operating System: Assembler (E) Programmer's Guide

IBM System/360 Operating System: Linkage Editor

Interpreting a TESTRAN Listing: Test information is printed on the system output device in a column 120 characters wide. Each page includes a standard page heading and an average of 55 lines of information produced by one or more TESTRAN statements. The general format of a page is shown by the sample page in Figure 12.

The circled numbers in Figure 12 distinguish five basic formats for individual lines of print. These are as follows:

1. ...TESTRAN OUTPUT... heads each page.
2. AT LOCATION... indicates entry to the TESTRAN interpreter at a test point.
3. ...MACRO ID... indicates one of the following:
  - Execution of a DUMP, TRACE, TEST OPEN, or TEST CLOSE statement.
  - Output resulting from an executed TRACE statement.
  - Detection of an error following execution of a statement.
4. EXECUTED STATEMENTS,... traces execution of GO, SET, TEST ON, and TEST WHEN statements.
5. \*\*\* IEGE... indicates a diagnostic message from the TESTRAN editor.

Each of these formats is described in detail in the remainder of this section.

The printing formats for specific types of data are shown in Table 1. The letters used to represent printing formats in the table are used with the same meanings throughout the remainder of this section. In addition, the letter y is used to designate a printed character for which the data type is variable.

```

① JOB1 TESTSTRAN OUTPUT DATE 10/164 TIME 10/04 PAGE 1
③ 1) MACRO ID 000, TEST OPEN , TESTSTRAN CONTROL SECTION = BEGIN , IDENTIFICATION JOB1
② AT LOCATION (SYMALTER) 0000EC 0100EC ENTER BEGIN
④ EXECUTED STATEMENTS, BEGIN 003
③ 2) MACRO ID 005, DUMP DATA STARTING IN SECTION SYMALTER
0154 INAREA
010154 COMEBACK MVC WRITAREA(88),ENTER CLEAR BUFFER FOR NEXT CARD 0003
③ 4) MACRO ID 006, DUMP PANEL
G*04* 00010154 G*08* 000100FC
PSW 00 0 1 0002 0 0 01008C CC=0 FIX POINT OVERFLOW OFF DEC OVERFLOW OFF EXP UNDERFLOW OFF SIGNIFICANCE OFF
④ EXECUTED STATEMENTS, BEGIN 007, 008
③ 1) MACRO ID 014, DUMP DATA STARTING IN SECTION SYMALTER
00FB ERRFLAG STARTIN STARTO
0100FB *
② AT LOCATION RETURN1 (SYMALTER) 0000DA 0100DA ENTER BEGIN
④ EXECUTED STATEMENTS, BEGIN 010
③ 3) MACRO ID 012, DUMP DATA STARTING IN SECTION SYMALTER
00FC OUTAREA
0100FC COMEBACK MVC WRITAREA(88),ENTER CLEAR BUFFER FOR NEXT CARD 0003
④ EXECUTED STATEMENTS, BEGIN 013
③ 1) MACRO ID 014, DUMP DATA STARTING IN SECTION SYMALTER
00FB ERRFLAG STARTIN STARTO
0100FB * 1
⑤ *** IEGE07 END OF TESTSTRAN EDIT--0000005 STATEMENTS PROCESSED

```

Figure 12. TESTSTRAN Editor Listing: Sample Page

Table 1. Printing Formats for Data Types

Data Type	Assumed Length in Bytes (1)	Printing Format (2)
Character (3)	1	c
Hexadecimal	1	xx
Binary	1	bbbbbbbb
Fixed-point (half-word)	2	sdddd (4)
Fixed-point (full-word)	4	sdddddddd (4)
Short floating-point	4	s0.dddddddd Esdd
Long floating-point	8	s0.dddddddddddd Esdd
Packed decimal	1	sd
Zoned decimal	1	sd
Address (5)		
Instruction:		
RR format	2	cccc xx
RS, RX, and SI formats	4	cccc xx x xxx
SS format	6	cccc xx x xxx x xxx

Notes to Table 1

1. The lengths assumed in definitions of printing formats are the assembler implied lengths for the corresponding data types. (Refer to Appendix A, Table 5.)
2. The letters shown in definitions of printing formats have the following meanings:

c is one EBCDIC character.  
x is one hexadecimal digit.  
b is one binary digit.  
s is an algebraic sign (+ or -).  
d is one decimal digit.  
0 is a high order zero.  
E means 'exponent'; the succeeding signed pair of digits is the exponent of the floating-point number.  
cccc is a machine mnemonic operation code.

3. Unprintable characters (other than blanks) are printed as two hexadecimal digits, the second of which appears on a separate line immediately below the first. For example, the hexadecimal data

C1D3D7C8C103C4C1E3C1

when edited into character format, is printed as

ALPHA0DATA

3

4. This format includes a decimal point that is positioned according to the scale factor associated with the data.
5. All addresses are printed in their source language formats.

PAGE HEADING (...TESTRAN OUTPUT...)

The following heading is printed at the top of each page:

```

-----
cccccccc          TESTRAN OUTPUT          DATE dd/ddd          TIME dd/dd          PAGE dddd
-----

```

cccccccc  
is the output identification specified as the third positional operand of the first-executed TEST OPEN statement.

DATE dd/ddd  
is the current date (year/day).

TIME dd/dd  
is the time (hour/minute) at which editing was begun.

PAGE dddd  
is the output page number.

TEST POINT IDENTIFICATION (AT LOCATION...)

The following line indicates entry to the TESTRAN interpreter at a test point:

```

-----
AT LOCATION ccccccc(ccccccc) xxxxxx xxxxxx ENTER ccccccc
-----

```

AT LOCATION ccccccc(ccccccc) xxxxxx xxxxxx  
identifies the test point. The field ccccccc(ccccccc) identifies the test point by name (if any), and by name (if any) of the control section that contains the test point. The fields xxxxxx xxxxxx are the assembled and loaded addresses of the test point.

ENTER ccccccc  
identifies the TESTRAN control section in which the test point was specified. (The control section is defined by an identically named TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

**Note:** The SELECT operand does not affect printing of the AT LOCATION line. This line is omitted, however, if it is not followed by the output of a DUMP or TRACE statement, or by an error message.

STATEMENT OUTPUT (...MACRO ID...)

Statement output is all output that is identified by "MACRO ID". It includes TEST OPEN, TEST CLOSE, DUMP and TRACE statement output, and error messages issued by the TESTRAN interpreter. Specific types of statement output are described below.

## DUMP CHANGES OUTPUT

DUMP CHANGES output is a change dump of main storage whose format is the same as that described below under "DUMP DATA Output."

## DUMP COMMENT OUTPUT

The following lines are a dump of a programmer-written comment.

```
d) MACRO ID ddd, DUMP COMMENT
cccc...
```

d)

is the class number assigned to the dump by a SELECT operand.

MACRO ID ddd, DUMP COMMENT

identifies the statement responsible for the dump. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09).

cccc...

is the dumped comment, which has a maximum length of 120 characters.

## DUMP DATA OUTPUT

The following lines are a dump of main storage:

```
d) MACRO ID ddd, DUMP DATA   STARTING IN SECTION cccccccc
   xxxx   cccccccc   cccccccc   cccccccc
   xxxxxx   YYYYYYYYYYYYYY... YYYYYYYYYYYYYY... YYYYYYYYYYYYYY...
```

d)

is the class number assigned to the dump by a SELECT operand.

MACRO ID ddd, DUMP CHANGES

identifies the statement responsible for the dump. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09).

STARTING IN SECTION cccccccc

identifies the control section that contains the dumped data.

xxxx  
xxxxxxx

are the assembled and loaded addresses of a dumped field. The field is the first field printed to the right of these addresses.

cccccccc

YYYYYYYYYYYYY...

are the symbolic name (if any) and contents of a dumped field. The name and format of the field are as defined in the problem program, or as specified by NAME and DATAM operands.

Note: The number of named fields per line varies from one to eleven due to differences in length; the starting positions are a minimum of nine printing positions apart. Fields too long for the current line are started on a new line.

In a dump of an instruction sequence, an instruction may be printed with the instruction SVC 26 immediately beneath it. If so, the instruction is located at a test point; the SVC instruction is the means by which the test point gives control to the TESTSTRAN interpreter. The SVC instruction replaced the original instruction when the test point was established; the original instruction was saved for execution on return of control to the test point.

DUMP MAP OUTPUT

The following lines are a map of control sections and allocated storage areas associated with a task that is active when a DUMP MAP statement is executed.

d) MACRO ID ddd, DUMP MAP						
NAME	TYPE	CSECT NAME	ASSEMBLED AT	LOADED AT	LENGTH-DEC	HEX
cccccccc	LOADED PROGRAM OBTAINED STORAGE	cccccccc	xxxxxxx	xxxxxxx xxxxxxx	dddd dddd	xxx xxx

d) is the class number assigned to the dump by a SELECT operand.

MACRO ID ddd, DUMP MAP identifies the statement responsible for the dump. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09).

NAME is a column heading. The column identifies each program (load module) associated with the active task (job step). Each program is represented by one line of print for each of its control sections. In a given line, the name ccccccc of a program is printed only if different from the name that applies to the previous line.

TYPE is a column heading. The column indicates the type of storage area that is represented. LOADED PROGRAM indicates a control section for which storage was reserved during assembly. OBTAINED STORAGE indicates an allocated storage area.

CSECT NAME is a column heading. The column identifies each control section of each program associated with the active task (job step).

ASSEMBLED AT is a column heading. The column contains the assembled address of each control section named in the dump.

LOADED AT is a column heading. The column contains the loaded address of each control section named in the dump. It also contains the address of each allocated storage area.

LENGTH-DEC HEX is a double column heading. The double column defines the decimal and hexadecimal length of each control section and allocated storage area.

Note: Some of the areas included in the dump will be areas allocated for use by the operating system.

#### DUMP PANEL OUTPUT

The following lines are a dump of registers and the program status word.

```
d) MACRO ID ddd, DUMP PANEL
G'dd' xxxxxxxx G'dd' xxxxxxxx
G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx G'dd' xxxxxxxx
PSW xx x x xxxx x x xxxxxx CC=d FIX POINT OVERFLOW ccc DEC OVERFLOW ccc EXP UNDERFLOW ccc SIGNIFICANCE ccc
F'dd' xxxxxxxx xxxxxxxx F'dd' xxxxxxxx xxxxxxxx F'dd' xxxxxxxx xxxxxxxx F'dd' xxxxxxxx xxxxxxxx
```

d) is the class number assigned to the dump by a SELECT operand.

MACRO ID ddd, DUMP PANEL identifies the statement responsible for the dump. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09).

G'dd' xxxxxxxx is the number (dd) and contents (xxxxxxx) of a dumped general register. The contents of the register are either in hexadecimal format as shown, or in some other format as specified by a DATAM operand.

PSW xx x xxxx x x xxxxxx is the program status word (PSW) stored on interruption of the problem program at the current test point.

CC=d specifies the value of the condition code (bits 34 and 35 of the program status word).

FIX POINT OVERFLOW ccc specifies the status of the fixed-point overflow mask (bit 36 of the program status word). The status ccc is either ON or OFF.

DEC OVERFLOW ccc specifies the status of the decimal overflow mask (bit 37 of the program status word). The status ccc is either ON or OFF.

EXP UNDERFLOW ccc specifies the status of the exponent underflow mask (bit 38 of the program status word). The status ccc is either ON or OFF.

SIGNIFICANCE ccc specifies the status of the significance mask (bit 39 of the program status word). The status ccc is either ON or OFF.

F'dd' xxxxxxxx xxxxxxxx is the number (dd) and contents (xxxxxxx xxxxxxxx) of a dumped floating-point register. The contents of the register are either in hexadecimal format as shown, or in some other format as specified by a DATAM operand.

## DUMP TABLE OUTPUT

The following lines are a dump of a system table (control block).

```
d) MACRO ID ddd, DUMP TABLE cccc cccccc BLOCK LOADED AT cccccc(ccccccc) xxxxxx xxxxxx
SECTION FIELD NAME CONTENTS
ccccccc
ccccccc yyyyy...
```

d)

is the class number assigned to the dump by a SELECT operand.

MACRO ID ddd, DUMP TABLE

identifies the statement responsible for the dump. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09).

cccc cccccc BLOCK

identifies the dumped table as a task control block, data control block, or data extent block.

LOADED AT cccccc(ccccccc) xxxxxx xxxxxx

specifies the location of a task control block or data control block. The field cccccc(ccccccc) specifies the name (if any) of a data control block and the name (if any) of the control section that contains the data control block. A single field xxxxxx specifies the address of a task control block; two fields xxxxxx xxxxxx specify both the assembled and loaded addresses of a data control block.

SECTION

is a column heading. The column identifies major sections of the table.

FIELD NAME

is a column heading. The column identifies fields within major sections of the table.

CONTENTS

is a column heading. The column defines the contents of each field.

## ERROR MESSAGE

The following lines indicate detection of an error during execution of a TESTRAN statement.

```
d) MACRO ID ddd, ERROR
*** IEGIdd cccc...
```

d)

is a class number assigned by a SELECT operand.

MACRO ID ddd, ERROR

identifies the statement that caused or detected the error. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09).

\*\*\* IEGIddd ccccc...  
is an error message issued by the TESTSTRAN interpreter. The text of the message (cccc...) is preceded by a standard system message code (IEGIddd). For an explanation of the message, refer to Appendix C, where all messages issued by the interpreter are listed in order by message code.

#### TEST CLOSE OUTPUT

The following lines indicate the execution of a TEST CLOSE statement.

```
d) MACRO ID ddd, TEST CLOSE  
ccccccc(ccccccc) xxxxxx xxxxxx ...
```

d)  
is the class number specified by the SELECT operand (if any) of a TEST OPEN statement.

MACRO ID ddd, TEST CLOSE  
identifies the TEST CLOSE statement. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09).

ccccccc(ccccccc) xxxxxx xxxxxx  
identifies a TESTSTRAN control section closed by the TEST CLOSE statement. The field ccccccc(ccccccc) contains a symbol generated during assembly and the name of the TESTSTRAN control section. The fields xxxxxx xxxxxx are the assembled and loaded addresses of the control section. (The control section is defined by an identically named TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

Note: The SELECT operand does not affect the printing of these lines of information.

#### TEST OPEN OUTPUT

The following lines indicate the execution of a TEST OPEN statement.

```
d) MACRO ID ddd, TEST OPEN , TESTSTRAN CONTROL SECTION = ccccccc, IDENTIFICATION ccccccc  
MAXIMUM NUMBER OF PAGES ddd, MAXIMUM NUMBER OF STATEMENTS ddd
```

d)  
is the class number specified by the SELECT operand (if any) of the TEST OPEN statement.

MACRO ID ddd, TEST OPEN , TESTSTRAN CONTROL SECTION = ccccccc  
identifies the TEST OPEN statement. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09). The name of the TESTSTRAN control section (ccccccc) is also the name of the TEST OPEN statement. (The control section is defined by the TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

IDENTIFICATION cccccccc  
specifies the output identification as provided by the third positional operand of the TEST OPEN statement.

MAXIMUM NUMBER OF PAGES ddd  
specifies the maximum number of pages produced.

MAXIMUM NUMBER OF STATEMENTS ddd  
specifies the maximum number of executed TESTRAN statements.

Note: The SELECT operand does not affect the printing of these lines of information.

#### TRACE CALL OUTPUT

The following groups of lines indicate the execution of a TRACE CALL statement and the later execution of a CALL macro-instruction.

```
d) MACRO ID ddd, TRACE CALL , cccccccc, FROM cccccccc(cccccccc) xxxxxx xxxxxx TO cccccccc(cccccccc) xxxxxx xxxxxx
STARTED
cccc...

d) MACRO ID ddd, TRACE CALL , cccccccc, TO cccccccc(cccccccc) xxxxxx xxxxxx AT cccccccc(cccccccc) xxxxxx xxxxxx
G'dd' xxxxxxxxx ...
G'dd' xxxxxxxxx ...
cccc...
```

d) is the class number assigned to the trace by a SELECT operand.

MACRO ID ddd, TRACE CALL , cccccccc,  
identifies the statement responsible for the trace. The identification number ddd is assigned by the assembler, and appears with the statement in the assembly listing (message number IEGM09). The field cccccccc is the name of the TESTRAN control section to which the statement belongs. (The control section is defined by an identically named TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

FROM cccccccc(cccccccc) xxxxxx xxxxxx TO cccccccc(cccccccc) xxxxxx xxxxxx  
defines the trace area. FROM cccccccc(cccccccc) specifies the name (if any) of the leftmost byte of the area and the name (if any) of the control section to which it belongs. TO cccccccc(cccccccc) gives the same information for the rightmost byte plus one. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.

TO cccccccc(cccccccc) xxxxxx xxxxxx AT cccccccc(cccccccc) xxxxxx xxxxxx  
identifies a called subroutine and the calling macro-instruction. TO cccccccc(cccccccc) specifies the name (if any) of the subroutine entry point, and the name (if any) of the control section that contains the entry point. FROM cccccccc(cccccccc) specifies the name (if any) of the CALL macro-instruction and the name (if any) of the control section that contains the CALL macro-instruction. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.

G'dd' xxxxxxxxx  
gives the number (dd) and contents (xxxxxxx) of a general register used by the CALL macro-instruction.

CCCCC...

is a comment specified by a COMMENT operand (if any) in the TRACE CALL statement. The maximum length is 120 characters.

#### TRACE FLOW OUTPUT

The following groups of lines indicate the execution of a TRACE FLOW statement and the later execution of a branch or SVC instruction.

```
-----  
d) MACRO ID ddd, TRACE FLOW , ccccccc, FROM ccccccc(cccccccc) xxxxxx xxxxxx TO ccccccc(cccccccc) xxxxxx xxxxxx  
STARTED  
cccccc...  
-----  
d) MACRO ID ddd, TRACE FLOW , ccccccc, FROM ccccccc(cccccccc) xxxxxx xxxxxx TO ccccccc(cccccccc) xxxxxx xxxxxx, CC=d  
cccc xx x xxx G'dd' xxxxxxxx ...  
cccccc...  
-----  
d) MACRO ID ddd, TRACE FLOW , ccccccc, FROM ccccccc(cccccccc) xxxxxx xxxxxx TO ccccccc(cccccccc) xxxxxx xxxxxx, CC=d  
cccc xx x xxx EXECUTED AS cccc xx x xxx BY EX xx x xxx FROM LOCATION ccccccc(cccccccc) xxxxxx xxxxxx  
G'dd' xxxxxxxx ...  
cccccc...  
-----
```

d) is the class number assigned to the trace by a SELECT operand.

MACRO ID ddd, TRACE FLOW , ccccccc,  
identifies the statement responsible for the trace. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09). The field ccccccc is the name of the TESTRAN control section to which the statement belongs. (The control section is defined by an identically named TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

FROM ccccccc(cccccccc) xxxxxx xxxxxx TO ccccccc(cccccccc) xxxxxx xxxxxx  
either (1) defines the trace area, or (2) identifies an executed branch or SVC instruction and the branch destination:

1. FROM ccccccc(cccccccc) specifies the name (if any) of the leftmost byte of the trace area, and the name (if any) of the control section to which it belongs. TO ccccccc(cccccccc) gives the same information for the rightmost byte plus one. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.
2. FROM ccccccc(cccccccc) specifies the name (if any) of an executed branch or SVC instruction, and the name (if any) of the control section that contains the instruction. TO ccccccc(cccccccc) specifies the name (if any) of the branch destination, and the name (if any) of the control section that contains the destination. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.

CC=d  
specifies the value of the condition code when the branch or SVC instruction is executed.

cccc xx x xxx  
is the branch or SVC instruction. (If an RR-type instruction, it is printed as cccc xx.)

EXECUTED AS cccc xx x xxx BY EX xx x xxx  
 indicates execution of a branch or SVC instruction by an EX instruction (EX xx x xxx). The instruction as executed is cccc xx x xxx (or cccc xx if it is an RR-type instruction). The effective values of bits 8-15 are shown.

FROM LOCATION ccccccc(cccccc) xxxxxx xxxxxx  
 specifies the location of the EX instruction. The field ccccccc(cccccc) specifies the name (if any) of the EX instruction and the name (if any) of the control section that contains the instruction. The fields xxxxxx xxxxxx are the assembled and loaded addresses of the EX instruction.

G"dd' xxxxxxxx  
 gives the number (dd) and contents (xxxxxxx) of a general register used by a branch or EX instruction.

CCCCC...  
 is a comment specified by a COMMENT operand (if any) in the TRACE FLOW statement. The maximum length is 120 characters.

TRACE REFER OUTPUT

The following groups of lines indicate the execution of a TRACE REFER statement and the later execution of a reference to data.

```

d) MACRO ID ddd, TRACE REFER , ccccccc, FROM ccccccc(cccccc) xxxxxx xxxxxx TO ccccccc(cccccc) xxxxxx xxxxxx
STARTED
cccc...

d) MACRO ID ddd, TRACE REFER , ccccccc, TO ccccccc(cccccc) xxxxxx xxxxxx FROM ccccccc(cccccc) xxxxxx xxxxxx
cccc xx x xxx x xxx G'dd' xxxxxxxx ...
cccc...
BEFORE yyyyyy...                                AFTER yyyyyy...

d) MACRO ID ddd, TRACE REFER , ccccccc, TO ccccccc(cccccc) xxxxxx xxxxxx FROM ccccccc(cccccc) xxxxxx xxxxxx
cccc xx x xxx x xxx EXECUTED AS cccc xx x xxx x xxx BY EX xx x xxx FROM LOCATION ccccccc(cccccc) xxxxxx xxxxxx
G'dd' xxxxxxxx ...
cccc...
BEFORE yyyyyy...                                AFTER yyyyyy...
  
```

d)  
 is the class number assigned to the trace by a SELECT operand.

MACRO ID ddd, TRACE REFER , ccccccc,  
 identifies the statement responsible for the trace. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09). The field ccccccc is the name of the TESTRAN control section to which the statement belongs. (The control section is defined by an identically named TEST OPEN statement, as indicated in the assembly listing by message number IEGM04.)

FROM ccccccc(cccccc) xxxxxx xxxxxx TO ccccccc(cccccc) xxxxxx xxxxxx  
 defines the trace area. FROM ccccccc(cccccc) specifies the name (if any) of the leftmost byte of the area and the name (if any) of the control section to which it belongs. TO ccccccc(cccccc) gives the same information for the rightmost byte plus one. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.

TO ccccccc(cccccc) xxxxxx xxxxxx FROM ccccccc(cccccc) xxxxxx xxxxxx identifies a field to which a reference is made and the instruction making the reference. TO ccccccc(cccccc) specifies the name (if any) of the field and the name (if any) of the control section that contains the field. FROM ccccccc(cccccc) specifies the name (if any) of the instruction making the reference and the name (if any) of the control section that contains the instruction. The fields xxxxxx xxxxxx are the corresponding assembled and loaded addresses.

cccc xx x xxx x xxx is the instruction making the reference. (If an RS-, RX-, or SI-type instruction, it is printed as cccc xx x xxx.)

EXECUTED AS cccc xx x xxx x xxx BY EX xx x xxx indicates that the instruction making the reference is executed by an EX instruction (EX xx x xxx). The instruction as executed is cccc xx x xxx x xxx (or cccc xx x xxx if it is an RS-, RX-, or SI-type instruction). The effective values of bits 8-15 are shown.

FROM LOCATION ccccccc(cccccc) xxxxxx xxxxxx specifies the location of the EX instruction. The field ccccccc(cccccc) specifies the name (if any) of the control section that contains the instruction. The fields xxxxxx xxxxxx are the assembled and loaded addresses of the EX instruction.

G'dd' xxxxxxxx gives the number (dd) and contents (xxxxxxx) of a general register used by the instruction making the reference, or by an EX instruction.

cccc... is a comment specified by a COMMENT operand (if any) in the TRACE REFER statement. The maximum length is 120 characters.

BEFORE yyyyy... specifies the contents of the field before the reference.

AFTER yyyyy... specifies the contents of the field after the reference.

#### TRACE STOP OUTPUT

The following line indicates execution of a TRACE STOP statement.

```
d) MACRO ID ddd, TRACE STOP ccccccc ddd, ddd, ..., ccccccc ddd, ddd, ...
```

d) is the class number assigned by a SELECT operand.

MACRO ID ddd, TRACE STOP identifies the TRACE STOP statement. The identification number ddd is assigned by the assembler and appears with the statement in the assembly listing (message number IEGM09).

ccccccc ddd, ddd,... identifies TRACE statements referred to by the TRACE STOP statement. Each number ddd is the identification number of a TRACE statement in a TESTRAN control section ccccccc. Each identification number is assigned by the assembler and appears with a statement in the

assembly listing (message number IEGM09). Each TESTRAN control section is defined by an identically named TEST OPEN statement (message number IEGM04).

Note: If the TRACE STOP statement does not refer to other statements by name, the word ALL is printed to indicate that all traces are stopped.

#### TESTRAN STATEMENT TRACE (EXECUTED STATEMENTS...)

The following line traces execution of GO, SET, TEST ON, and TEST WHEN statements.

```
EXECUTED STATEMENTS, ccccccc ddd, ddd, ..., ccccccc ddd, ddd, ...
```

ccccccc ddd, ddd, ... identifies the executed statements. Each number ddd is the identification number of a statement in a TESTRAN control section named ccccccc. Each identification number is assigned by the assembler and appears with a statement in the assembly listing (message number IEGM09). Each TESTRAN control section is defined by an identically named TEST OPEN statement (message number IEGM04).

Note: This line is printed only if followed by the output of a DUMP or TRACE statement, or by an error message. The number of statements identified is limited to 28: the first 27 statements that are executed and the last statement that is executed before other output is generated.

#### TESTRAN EDITOR MESSAGE (\*\* IEGE...)

The following line is a diagnostic message issued by the TESTRAN editor.

```
** IEGEd d ccccc...
```

IEGEdd is a message code that identifies the message.

cccccc... is the message text. For an explanation of the text, refer to Appendix C, which lists all messages in order by message code.

This appendix formally describes the function and format of TESTRAN statements.

CODING CONVENTIONS

TESTRAN statements are coded according to the coding conventions for assembler language macro-instructions. These conventions are described in the publication IBM System/360 Operating System: Assembler Language. They should be familiar to the reader who has experience in writing his own macro-instructions or in using those defined by the system for requesting supervisor and data management services.

The coding of macro-instructions differs in two ways from the coding of machine instructions:

1. There is no limit to the number of continuation lines that can be used.
2. There is a wider variety of operands.

For the reader who has no experience with macro-instructions, the following brief description of the operand field may be helpful.

The Operand Field: As in a machine instruction, the operand field consists of individual operands that are separated by commas. The meaning of each operand either is implied by its position in the field or is expressed by a keyword that is part of the operand itself. For example, the three statements

```
DUMP DATA,CHANGES,DATAM=L8,SELECT=1
DUMP DATA,CHANGES,SELECT=1,DATAM=L8
DUMP CHANGES,DATA,SELECT=1,DATAM=L8
```

each contain two positional operands followed by two keyword operands. Because the position of a keyword operand is unimportant, the first two statements are functionally equivalent; they are not equivalent to the third statement, which differs in its first and second (positional) operands.

Some operands are optional: they can be written or omitted as the programmer chooses. If a positional operand is omitted, it must be represented by a comma if it precedes a positional operand that is not omitted. In the statement

```
TEST ON,,,2,EVEN
```

the second and third operands are omitted and each is represented by a comma. An omitted positional operand is not represented by a comma if it is not followed by a positional operand that is actually written. An omitted keyword operand is never represented by a comma.

To allow the use of commas within an operand, a positional operand or the right-hand part of a keyword operand can sometimes be enclosed in parentheses. Within the parentheses, commas separate individual items of information, which together are called a sublist. In the statements

```
T1 TRACE STOP,(TRACE#1,TRACE#2,TRACE#3)
D1 DUMP DATA,INPUT,INPUT+80,DSECT=(INPUT,3)
```

the second (positional) operand of T1 is a sublist of three items, and the DSECT keyword operand in D1 contains a sublist of two items. The number of items in a sublist is variable; the programmer specifies one or more items as he chooses. If only one item is specified, no commas are needed to separate items and the enclosing parentheses can be omitted.

FUNCTIONS OF TESTRAN STATEMENTS

The following pages describe the functions of TESTRAN statements and their operands. For convenience, the statements are divided into four groups:

- DUMP and TRACE statements.
- TEST statements.
- GO statements.
- SET statements.

The description of each group has two parts:

- A list of the statements in the group and the general function of each statement.
- A list of the operands for the statements and the specific function of each operand.

To use this part of the appendix, turn to the last part, "Format of TESTRAN Statements," and fold out the last page to show Tables 2-5. Table 2 defines the format of TESTRAN statements using conventions that are standard in the Systems Reference Library. Tables 3-5 present supplementary

information about the format of certain operands.

To write a statement using this appendix, first select a statement from the list

of statements for one of the groups. Refer to the tables on the foldout page to learn the format of the statement. Then refer to the list of operands for a description of each operand indicated in the tables.

---

## DUMP AND TRACE STATEMENTS

The DUMP and TRACE statements record information about the problem program. Their basic functions are as follows:

- DUMP DATA**  
dumps a storage area.
- DUMP CHANGES**  
dumps changes to a storage area; dumped fields are printed only if (1) contained in the first dump taken by the statement, or (2) contained in a later dump and changed since the previous dump by the same statement.
- DUMP COMMENT**  
dumps a programmer's comment contained in the statement.
- DUMP MAP**  
dumps a map of control sections and allocated storage areas associated with the active task (job step).
- DUMP PANEL**  
dumps general and floating-point registers and the program status word stored at the most recent interruption of the problem program.
- DUMP TABLE**  
dumps a specified system table (control block).
- TRACE CALL**  
traces subroutine calls by CALL macro-instructions in a specified storage area.
- TRACE FLOW**  
traces transfers (by branch and SVC instructions) to, from, or within a storage area.
- TRACE REFER**  
traces references to a storage area by instructions that could change data within that area.
- TRACE STOP**  
stops traces started by TRACE CALL, TRACE FLOW, and TRACE REFER statements.

Operands: The operands of the DUMP and TRACE statements are as follows:

address

- as the second operand, points to the leftmost byte of a storage area.
- as the third operand, points to the rightmost byte plus one of a storage area, with this exception: in a DUMP TABLE statement, the third operand points to the data control block (DCB) that is dumped or that is associated with the data extent block (DEB) that is dumped.

Note on Storage Areas: A storage area is defined by the effective values of the address operands at the time a DUMP or TRACE statement is executed. Indexing of addresses may cause an area to vary in size and location when a statement is executed several times (i.e., at several test points). A change dump of a variable area includes all additions to the previously dumped area, plus changed data that lies within both the present and previous areas. A trace is shifted from the previously defined area to a newly defined area on each execution of the statement that first started the trace.

If a statement does not point to both ends of a storage area, the length of the area is determined by the DATAM keyword operand. If this operand is omitted, the length is determined by the length attribute of the first symbol used in the address. If the address contains no symbols, or only an external symbol, the length is assumed to be one byte.

'text'

specifies a programmer's comment.

(registers[,registers]...)

- specifies the registers to be dumped.
- if absent, implies that all registers are to be dumped.

Note on Printing Format: Unless the DATAM keyword operand is written,

dumped registers (including floating-point registers) are printed in hexadecimal format.

TCB|DCB|DEB

specifies the system table to be dumped, as follows:

- TCB - the task control block for the active task.
- DCB - an open data control block whose address is the third operand of the statement.
- DEB - the data extent block for an open data control block whose address is the third operand of the statement.

(symbol[,symbol]...)

- specifies the names of one or more TRACE statements that started traces which are to be stopped.
- if absent, implies that all current traces are to be stopped.

d& DSECT=(symbol{,1|integer})  
identifies a storage area as a dummy control section, or as part of a dummy control section.

symbol

1. is the name of the dummy control section.
2. specifies the assumed location of the dummy control section, which must be addressable by means of a base register previously defined and loaded by the problem program.

1|integer

1. specifies a number by which the length of the storage area is multiplied on execution of the statement.

2. specifies the maximum number of times the format of the dummy control section is to be repeated to define the format of printed information.

d DATAM=[type][L{length}][S{scale}]

- specifies type, length, and scale attributes.
- determines either the length of a storage area when the third positional operand is omitted, or the length of data (right justified) in a dumped register.
- determines either the printing format for each field of a storage area when the DSECT operand is omitted, or the printing format of data dumped from a register.

n NAME=symbol

- provides a symbol to be printed as the name of the first field of a dump.
- suppresses the printing of field names as they are defined in the source program.

c COMMENT='text'

annotates trace output with a programmer's comment.

Δ SELECT={1|2|3|4|5|6|7|8}

- classifies test information produced by the DUMP or TRACE statement; reclassifies this information if it has already been classified in a TEST OPEN or TEST AT statement.
- identifies the class by a number, which can be used (in a job control statement) to select the class for printing by the TESTSTRAN editor.

## TEST STATEMENTS

Test statements are of three types:

### Linkage statements

- TEST OPEN
- TEST CLOSE

### Specification statements

- TEST AT
- TEST DEFINE

### Decision-making statements

- TEST ON
- TEST WHEN

Each type is described separately.

When executed, the TEST OPEN statement "opens" the control section by reference to TEST specification statements. It establishes a link to the TESTSTRAN interpreter at each test point specified in a TEST AT statement, and it creates counters and flags as specified in TEST DEFINE statements.

A TEST CLOSE statement closes a TESTSTRAN control section by nullifying the linkages, counters, and flags established when the control section was opened. When closed, a control section cannot be entered by a branch from another TESTSTRAN control section, and its counters and flags cannot be used by statements in other control sections.

Operands: The operands of the TEST linkage statements are as follows:

### Linkage Statements

The TEST OPEN and TEST CLOSE statements control linkage between the problem program and the TESTSTRAN interpreter. Their basic functions are as follows:

#### TEST OPEN

- defines a TESTSTRAN control section having the same name as the statement itself.
- opens the TESTSTRAN control section; loads the standard entry point register (15), and passes control to the problem program entry point (second operand).
- specifies task options (third, fourth, MAXE, and MAXP operands).
- chains the opening of other TESTSTRAN control sections (OPTEST operand).
- classifies test information for selective retrieval (SELECT operand).

#### TEST CLOSE

- closes the TESTSTRAN control section in which it is located.
- closes any other TESTSTRAN control sections that were opened at the same time by chained opening.
- returns control to the problem program.

A TEST OPEN statement defines a control section that includes all TESTSTRAN statements that precede the next TEST OPEN statement, if any, in the source program. It must be the first TESTSTRAN statement in the source program.

#### address

- is placed in register 15.
- receives control after TEST OPEN.
- is required if TEST OPEN is executed.
- is ignored if TEST OPEN is not executed (i.e., if opening of the control section is chained by the execution of another TEST OPEN statement).

#### task options

- control testing under a task (job step) and editing of the resulting test output.
- can be specified only in the first TEST OPEN statement executed under a task; are ignored when specified in other TEST OPEN statements.
- are specified by four operands, as follows:

#### symbol

1. appears in the heading of each page printed by the TESTSTRAN editor.
2. identifies test information produced under the task.

#### LINK|LOAD

1. specifies which system macro-instruction the TESTSTRAN interpreter should use to load its nonresident modules.
2. reduces (LINK) or increases (LOAD) both the storage requirements and the operating speed of the interpreter.

MAXE=integer

1. specifies the maximum number of statements to be executed by the TESTRAN interpreter.<sup>1</sup>
2. causes abnormal termination if the maximum is exceeded.

MAXP=integer

1. specifies the maximum number of pages of test information to be produced.<sup>1</sup>
2. causes abnormal termination if the maximum is exceeded.

OPTEST=(symbol[,symbol],...)

- points to the TESTRAN control sections defined by other TEST OPEN statements.
- chains the opening of these control sections: causes all of them to be opened when the TEST OPEN statement is executed.
- is ignored if the TEST OPEN statement is not executed.

4 SELECT={1|2|3|4|5|6|7|8}

- classifies test information produced by control sections opened by the TEST OPEN statement.
- identifies the class by a number, which can be used (in a job control statement) to select the class for printing by the TESTRAN editor.
- is ignored if the TEST OPEN statement is not executed.

### Specification Statements

The TEST AT and TEST DEFINE statements specify functions that are performed when the TESTRAN control section is opened:

TEST AT

- specifies one or more test points in the problem program (second operand).
- classifies test information for selective retrieval (SELECT operand).

TEST DEFINE

defines TESTRAN counters or flags.

<sup>1</sup>This maximum must not exceed the installation maximum established during system generation. If it does, or if the operand is omitted, the installation maximum is assumed.

A TEST AT statement must be placed so that the next sequential TESTRAN statement is the first that should be executed at each specified test point. A TEST DEFINE statement can be placed anywhere in a TESTRAN control section.

In an executed sequence of TESTRAN statements, a TEST AT statement returns control to the problem program. A TEST DEFINE statement performs no operation; the next sequential statement is executed.

Operands: The operands of the TEST specification statements are as follows:

([\*|address][,address]...)

- points to one or more test points in the problem program.
- refers, if \* is written, to the value of the location counter for the current problem program control section.
- is subject to the following restrictions:
  1. Each specified address must be that of an instruction in the problem program.
  2. The instruction must not be a privileged or SVC instruction, or an EX instruction that executes a privileged or SVC instruction.
  3. The instruction must not be modified by any instruction or executed by an EX instruction.

4 SELECT={1|2|3|4|5|6|7|8}

- classifies test information recorded at test points specified in the statement; reclassifies this information if it has already been classified in a TEST OPEN statement.
- identifies the class by a number which can be used (in a job control statement) to select the class for printing by the TESTRAN editor.

COUNTER|FLAG

determines whether counters or flags are defined by the statement.

Note on Counters and Flags: A counter is a full-word, fixed-point value. A flag is a single binary digit. Both are set to zero when the control section is opened. Their values are lost when the control section is closed.

(symbol[,symbol]...)

specifies a name for each counter or flag (the number of names determines the number of counters or flags that are defined).

## Decision-Making Statements

The TEST ON and TEST WHEN statements perform decision-making functions based on conditional branching to other TESTSTRAN statements. Their functions are as follows:

### TEST ON

- increments a counter (COUNTER operand) by one.
- tests the counter against three values (second, third, and fourth operands).
- branches to a TESTSTRAN statement (fifth operand) if the value of the counter (1) is greater than or equal to the second operand, (2) is less than or equal to the third operand, and (3) is a multiple of the fourth operand.

### TEST WHEN (first form)

- tests the value of a flag (second operand).
- branches to a TESTSTRAN statement (third operand) if the value of the flag is one.

### TEST WHEN (second form)

- specifies a logical relationship between two flags (second, third and fourth operands).
- branches to a TESTSTRAN statement (fifth operand) if the relationship is correct.

### TEST WHEN (third form)

- specifies an arithmetic relationship between counters and/or variables (second, third, and fourth operands).
- branches to a TESTSTRAN statement (fifth operand) if the relationship is correct.

Operands: The operands of the TEST ON and TEST WHEN statements are as follows:

*ari* address|register|integer

- specifies a fixed-point value from 1 to  $2^{31}-1$ .

address

1. points to a full-word, fixed-point value in main storage; this value need not be on a full-word boundary.
2. cannot be written as an integer.

register

points to a full-word, fixed-point value in a general register.

integer

is a decimal value that is assembled as a full-word, fixed-point value.

- if absent, implies the value 1 for the second or fourth operand, or the value  $2^{31}-1$  for the third operand.

symbol

- as the second or fourth operand, points to a flag defined by a TEST DEFINE statement.
- as the third or fifth operand, points to a TESTSTRAN statement.

COUNTER=symbol

- points to a counter defined by a TEST DEFINE statement.
- if absent, implies that the statement increments and tests an unnamed counter that is automatically defined for exclusive use by the statement.

AND|OR

specifies a logical relationship between the flags specified by the second and fourth operands.

AND

specifies that the value of both flags is one.

OR

specifies that the value of one flag, or of both, is one.

*arl* address|register|literal  
specifies an arithmetic value.

address

points to a value in the problem program or to a TESTSTRAN counter.

register

points to a value in a register.

literal

specifies a constant value that is assembled as part of the statement.

Note on Data Format: If the DATAM operand is omitted, the format (type and length) of both the second and fourth operands is implied as follows:

- If the second operand is an address, the format is determined by the attributes of the first symbol used in the address. If the address contains no symbols, or only an external symbol, the format is determined by the fourth operand in the same manner as by the second operand. However, if the fourth operand is also an address, and contains no symbol other than an external symbol, the format is assumed to be hexadecimal with a length of one byte.
- If the second operand is a reference to a general register, the format is hexadecimal with a length of four bytes. If it is a reference to a floating-point register, the format is floating-point with a length of eight bytes.

- If the second operand is a literal, the format is specified or implied by the literal notation.

GT|GE|EQ|NE|LT|LE

- specifies an arithmetic relationship between the values specified by the second and fourth operands.
- has the following meaning:

GT - greater than  
 GE - greater than or equal to  
 EQ - equal to  
 NE - not equal to  
 LT - less than  
 LE - less than or equal to

d DATAM=[type][L{length}][S{scale}]

- specifies type, length, and scale attributes.
- defines the type and length of values specified by the second and fourth operands.

## GO STATEMENTS

The GO statements provide branching functions. Their basic functions are as follows:

- GO TO  
 branches unconditionally to a TESTRAN statement.
- GO IN  
 calls an internal subroutine.
- GO OUT  
 returns from an internal subroutine.
- GO BACK  
 returns control to the problem program, or passes control to a specified executable instruction.

Operands: The operands of the GO statements are as follows:

symbol

is the name of the TESTRAN statement that is branched to or that is the first statement of an internal subroutine.

Note on Internal Subroutines: A maximum of three levels of internal subroutines can be maintained. The first level is lost if a fourth level is created.

address

- points to an executable instruction to which control is passed.
- if absent, causes execution of the problem program instruction at the current test point.

## SET STATEMENTS

The SET statements assign values to counters, flags, and variables. Their functions are as follows:

**SET COUNTER**  
assigns a value to a TESTRAN counter.

**SET FLAG**  
assigns a value to a TESTRAN flag.

**SET VARIABLE**  
assigns a value to a problem program variable (storage field or register).

**Operands:** The second operand of each SET statement points to a counter, flag, or variable; the third operand specifies the value that is assigned. The operands are as follows:

### symbol

- in a SET COUNTER statement, points to a TESTRAN counter.
- in a SET FLAG statement, points to a TESTRAN flag.

*arl* address|register|literal  
specifies the new value of a counter or variable.

address  
points to a value in the problem program or to a TESTRAN counter.

register  
points to a value in a register.

literal  
specifies a constant value that is assembled as part of the statement.

=0|=1  
specifies the new value (zero or one) of a TESTRAN flag.

address|register  
points to a variable to which a value is assigned.

address  
points to a value in the problem program.

register  
points to a value in a register.

**Note on Data Format:** If the DATAM operand is omitted, the length of the values specified by the second and third operands is determined as follows:

- If the second operand is an address, the length is determined by the first symbol used in the address. If the address contains no symbols, or only an external symbol, the length is determined by the third operand in the same manner as by the second operand. However, if the third operand is also an address and contains no symbol other than an external symbol, the length is assumed to be one byte. If the third operand is a literal, the length is specified or implied by the literal notation.
- If the second operand is a reference to a general register, the implied length is four bytes. If it is a reference to a floating-point register, the implied length is eight bytes.

*d* DATAM=[type][L{length}][S{scale}]  
• specifies type, length, and scale attributes.  
• defines the length of values specified by the second and third operands of a SET VARIABLE statement.

FORMAT OF TESTRAN STATEMENTS

The format of TESTRAN statements is defined in Table 2. The following conventions are used:

- Uppercase letters, numbers, and punctuation marks must be coded by the programmer exactly as shown. Exceptions to this convention are brackets [ ], the vertical stroke |, braces { }, the ellipsis ... and superscripts. These are never coded.
- Lowercase letters and words represent variables for which specific information or specific values must be substituted by the programmer when coding. Their meanings are given in Tables 3 and 4. Table 3 defines abbreviations, which are shown in italics; Table 4 defines basic variables used in both Table 2 and Table 3.

- Items or groups of items within brackets [ ] are optional. They may be omitted at the programmer's discretion.
- Items separated by a vertical stroke | are alternatives. No more than one of the alternative items should be coded by the programmer.
- If an alternative item is underlined, that item is implied; that is, the operating system will automatically assume it is the programmer's choice when none of the items is coded.
- Braces { } are used to group related items, such as several alternative items.
- An ellipsis ... indicates that the preceding item or group of items can be coded more than once in succession.
- A superscript number refers to a footnote to the description of a statement.

Table 2. Format of TESTRAN Statements

Name	Operation	Operand
[symbol]	DUMP	{DATA CHANGES}, address[,address][,ds][,d][,n][,s] COMMENT, 'text'[,s] MAP[,s] PANEL[(registers[,registers]...)][,d][,s] TABLE,{TCB {DCB DEB},address}[,s]
[symbol]	TRACE	CALL, address, address[,c][,s] FLOW, address[,address][,c][,s] REFER, address[,address][,ds][,d][,c][,s] STOP[(symbol[,symbol]...)][,s]
symbol <sup>1</sup>	TEST	OPEN[,address <sup>2</sup> [,task options][,OPTEST=(symbol[,symbol]...)][,s] CLOSE AT, (* address <sup>2</sup> [,address <sup>2</sup> ]...)[,s] DEFINE, {COUNTER FLAG}, (symbol[,symbol]...) ON, [ari], [ari], [ari], symbol[,COUNTER=symbol] WHEN, symbol, symbol WHEN, symbol, {AND OR}, symbol, symbol WHEN, arl, {GT GE EQ NE LE LT}, arl, symbol[,s]
[symbol]	GO	{TO IN}, symbol OUT BACK[,address]
[symbol]	SET	COUNTER, symbol, arl FLAG, symbol, {symbol =0 =1} VARIABLE, {address register}, arl[,d]

<sup>1</sup>A symbol is required in the name field of a TEST OPEN statement; it is optional in other TEST statements.  
<sup>2</sup>This operand can be written only as a nonindexed implied address.

Table 3. Definition of Abbreviations

Abbreviation	Definition
<i>ari</i>	address
<i>arl</i>	address
<i>c</i>	COMPL
<i>d</i>	DAT
<i>ds</i>	DSE
<i>n</i>	NAM
<i>s</i>	SEL
<i>task options</i>	[sy]

Table 4. Definition of Variables

Variable	Definition
address	An
integer	A d
length	An
literal	A c
register	A p
registers	A p
scale	A s
symbol	A s
text	A c
type	A s

<sup>1</sup>The format of a

Imp  
Exp

s i  
exp  
reg  
add

An implied ad  
appears in the  
constant. If  
contents of the

<sup>2</sup>The format of t

Gen  
Flo

reg  
val  
a s

<sup>3</sup>The format of t  
statement that  
is enclosed by  
must include a  
or ampersand  
apostrophes or

Table 3. Definitions of Abbreviations Used in Table 2

Abbreviation	Definition
ari	address register integer
arl	address register literal
c	COMMENT='text'
d	DATAM=[type][L{length}][S{scale}]
ds	DSECT=(symbol{,1},integer)
n	NAME=symbol
s	SELECT={1 2 3 4 5 6 7 8}
task options	{symbol}{,LINK LOAD}{,MAXE=integer}{,MAXP=integer}

Table 4. Definitions of Variables Used in Tables 2 and 3

Variable	Definition
address	An indexed or nonindexed implied or explicit address <sup>1</sup>
integer	A decimal self-defining term
length	An unsigned decimal integer (see Table 5)
literal	A constant preceded by an equal sign (=)
register	A pointer to a general or floating-point register <sup>2</sup>
registers	A pointer to one or a series of general or floating-point registers <sup>2</sup>
scale	A signed or unsigned decimal integer (see Table 5)
symbol	A string of letters and digits that begins with a letter and is not longer than eight characters
text	A character constant <sup>3</sup>
type	A standard data type code (see Table 5)

<sup>1</sup>The format of an address is given by the following table:

	Indexed	Nonindexed
Implied address	s(x)	s
Explicit address	d(x,b)	d(0,b)

s is an absolute or relocatable expression; d, x, and b are absolute expressions. s is a numeric or symbolic storage address; x is an index register number; d is a displacement from a base address; b is a base address register number.

An implied address is assembled in base-displacement form only if a DSECT operand appears in the same statement. Normally, it is evaluated by an A-type address constant. If it is indexed, its effective value is that of the constant plus the contents of the index register at the time the statement is executed.

<sup>2</sup>The format of the pointer is given by the following table:

	Single Register	Series of Registers
General register	G'reg'	G'reg <sub>1</sub> ,reg <sub>n</sub> '
Floating-point register	F'reg'	F'reg <sub>1</sub> ,reg <sub>n</sub> '

reg, reg<sub>1</sub>, and reg<sub>n</sub> are each a symbol or decimal integer whose value is a valid register number. reg<sub>1</sub> and reg<sub>n</sub> are the first and last registers of a series. reg<sub>n</sub> can have either a higher or a lower value than reg<sub>1</sub>.

<sup>3</sup>The format of the character constant is that of the constant subfield of a DC statement that defines a character constant. As shown in Tables 2 and 3, the constant is enclosed by apostrophes. The constant can include any valid EBCDIC character, but must include a pair of apostrophes or ampersands to represent a single apostrophe (') or ampersand (&). The maximum length is 120 characters, counting each pair of apostrophes or ampersands as a single character.

Table 5. Definition of Type, Length, and Scale

Type		Length in bytes <sup>1</sup>		Scale <sup>2</sup>
Code	Meaning	Specified	Implied	
C	character	1 to 256	1	(not applicable)
X	hexadecimal	1 to 256	1	(not applicable)
B	binary	1 to 256	1	(not applicable)
H	fixed-point	1 to 8	2	-187 to +346
F	fixed-point	1 to 8	4	-187 to +346
E	floating-point	1 to 8	4	(not applicable)
D	floating-point	1 to 8	8	(not applicable)
P	packed decimal	1 to 16	1	(not applicable)
Z	zoned decimal	1 to 16	1	(not applicable)
I	instruction	(not applicable)	(variable)	(not applicable)

<sup>1</sup>The implied length is used if the type, but not the length, is specified.

<sup>2</sup>The implied scale is zero if no scale is specified. If a positive scale is intended, the sign (+) can be omitted.

APPENDIX B: IBM-SUPPLIED CATALOGED PROCEDURES

This appendix defines cataloged procedures that are supplied by IBM and are referred to in Section 3 of this publication.

PROCEDURE ASMEC

```
//ASM      EXEC  PGM=IETASM                      00020000
//SYSLIB   DD    DSNAME=SYS1.MACLIB,DISP=OLD      00040000
//SYSUT1   DD    UNIT=SYSSQ,SPACE=(400,(400,50)) 00060000
//SYSUT2   DD    UNIT=SYSSQ,SPACE=(400,(400,50)) 00080000
//SYSUT3   DD    UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//          SPACE=(400,(400,50))                C00100000
//          UNIT=SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//          SPACE=(400,(400,50))                00120000
//SYSPRINT DD    SYSOUT=A                        00140000
//SYSPUNCH DD    UNIT=SYSCP                      00160000
```

PROCEDURE LKED

```
//LKED     EXEC  PGM=IEWL,PARM='XREF,LIST,LET,NCAL' 00020000
//SYSPRINT DD    SYSOUT=A                        00040000
//SYSLIN   DD    DDNAME=SYSIN                   00060000
//SYSLMOD  DD    DSNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),
//          UNIT=SYSDA,DISP=(MOD,PASS)          C00080000
//          UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//          SPACE=(1024,(200,20))              C00120000
//          SPACE=(1024,(200,20))              00140000
```

PROCEDURE TASME

```
//ASM      EXEC  PGM=IETASM,PARM=(LOAD,TEST)      00020000
//SYSLIB   DD    DSNAME=SYS1.MACLIB,DISP=(OLD)    00040000
//SYSUT1   DD    UNIT=(SYSSQ,SEP=(SYSLIB)),
//          SPACE=(400,(150,20))                C00060000
//          SPACE=(400,(150,20))                00080000
//SYSUT2   DD    UNIT=(SYSSQ,SEP=(SYSUT1)),SPACE=(400,(150,20)) 00100000
//SYSUT3   DD    UNIT=(SYSSQ,SEP=(SYSLIB,SYSUT2)),
//          SPACE=(400,(150,20))                C00120000
//          SPACE=(400,(150,20))                00140000
//SYSPRINT DD    SYSOUT=A                        00160000
//SYSPUNCH DD    DSNAME=&LOADSET,UNIT=SYSDA,
//          SPACE=(80,(200,50)),DISP=(MOD,PASS) C00180000
//          SPACE=(80,(200,50)),DISP=(MOD,PASS) 00200000
//LKED     EXEC  PGM=IEWL,PARM=(XREF,LIST,LET,NCAL,TEST) 00220000
//SYSPRINT DD    SYSOUT=A                        00240000
//SYSLIN   DD    DSNAME=&LOADSET,DISP=(OLD)      00260000
//          DD    DDNAME=SYSIN                   00280000
//SYSLMOD  DD    DSNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),
//          UNIT=SYSDA,DISP=(MOD,PASS)          C00300000
//          UNIT=(SYSDA,DISP=(MOD,PASS))        00320000
//SYSUT1   DD    UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//          SPACE=(1024,(200,20))              C00340000
//          SPACE=(1024,(200,20))              00360000
```

PROCEDURE TASMEG

```
//ASM      EXEC   PGM=IETASM,PARM=(LOAD,TEST)                00020000
//SYSLIB   DD     DSNAME=SYS1.MACLIB,DISP=(OLD)                00040000
//SYSUT1   DD     UNIT=(SYSSQ,SEP=(SYSLIB)),                    C00060000
//          SPACE=(400,(150,20))                              00080000
//SYSUT2   DD     UNIT=(SYSSQ,SEP=(SYSUT1)),SPACE=(400,(150,20)) 00100000
//SYSUT3   DD     UNIT=(SYSSQ,SEP=(SYSLIB,SYSUT2)),            C00120000
//          SPACE=(400,(150,20))                              00140000
//SYSPRINT DD     SYSOUT=A                                     00160000
//SYSPUNCH DD     DSNAME=&LOADSET,UNIT=SYSDA,                  C00180000
//          SPACE=(80,(200,50)),DISP=(MOD,PASS)                00200000
//LKED     EXEC   PGM=IEWL,PARM=(XREF,LIST,LET,NCAL,TEST)      00220000
//SYSPRINT DD     SYSOUT=A                                     00240000
//SYSLIN   DD     DSNAME=&LOADSET,DISP=(OLD)                    00260000
//          DD     DDNAME=SYSIN                                00280000
//SYSLMOD  DD     DSNAME=&GOSET(GO),SPACE=(1024,(50(20(1))),    C00300000
//          UNIT=SYSDA,DISP=(MOD,PASS)                          00320000
//SYSUT1   DD     UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),            C00340000
//          SPACE=(1024,(200,20))                              00360000
//GO       EXEC   PGM=*.LKED.SYSLMOD                          00380000
```

PROCEDURE TASMEGED

```
//ASM      EXEC   PGM=IETASM,PARM=(LOAD,TEST)                00020000
//SYSLIB   DD     DSNAME=SYS1.MACLIB,DISP=(OLD)                00040000
//SYSUT1   DD     UNIT=(SYSSQ,SEP=(SYSLIB)),                    C00060000
//          SPACE=(400,(150,20))                              00080000
//SYSUT2   DD     UNIT=(SYSSQ,SEP=(SYSUT1)),SPACE=(400,(150,20)) 00100000
//SYSUT3   DD     UNIT=(SYSSQ,SEP=(SYSLIB,SYSUT2)),            C00120000
//          SPACE=(400,(150,20))                              00140000
//SYSPRINT DD     SYSOUT=A                                     00160000
//SYSPUNCH DD     DSNAME=&LOADSET,UNIT=SYSDA,                  C00180000
//          SPACE=(80,(200,50)),DISP=(MOD,PASS)                00200000
//LKED     EXEC   PGM=IEWL,PARM=(XREF,LIST,LET,NCAL,TEST)      00220000
//SYSPRINT DD     SYSOUT=A                                     00240000
//SYSLIN   DD     DSNAME=&LOADSET,DISP=(OLD)                    00260000
//          DD     DDNAME=SYSIN                                00280000
//SYSLMOD  DD     DSNAME=&GOSET(GO),SPACE=(1024,(50,20,1)),    C00300000
//          UNIT=SYSDA,DISP=(MOD,PASS)                          00320000
//SYSUT1   DD     UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),            C00340000
//          SPACE=(1024,(200,20))                              00360000
//GO       EXEC   PGM=*.LKED.SYSLMOD                          00380000
//SYSTEM   DD     DSNAME=&TESTSET,SPACE=(300,(100)),            C00400000
//          UNIT=SYSSQ,DISP=(NEW,PASS)                          00420000
//EDIT     EXEC   PGM=IEGTTEDT                                 00440000
//SYSUT1   DD     UNIT=SYSDA,SPACE=(500,(100))                 00460000
//SYSTEM   DD     DSNAME=&TESTSET,UNIT=(SYSSQ,SEP=(SYSUT1)),    C00480000
//          DISP=(OLD,DELETE)                                  00500000
//SYSPRINT DD     SYSOUT=A                                     00520000
```

PROCEDURE TTED

```
//EDIT     EXEC   PGM=IEGTTEDT                                 00020000
//SYSUT1   DD     UNIT=SYSDA,SPACE=(500,(100))                 00040000
//SYSPRINT DD     SYSOUT=A                                     00060000
```

This appendix reproduces the following sections from the publication IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps:

- TESTRAN Editor Messages
- TESTRAN Interpreter Messages
- TESTRAN Macro-Expansion Messages

Table 6 describes the messages in these sections.

Table 6. TESTRAN Messages

MESSAGE	WHERE PRINTED	MESSAGE FORMAT	COMMENTS
TESTRAN Editor Messages	TESTRAN Listing (TESTRAN editor SYSPRINT data set)	<p>*** IEGEnn text</p> <p>IEG = TESTRAN message code</p> <p>Enn = Message serial number indicating the TESTRAN editor</p> <p>text = Message text</p>	Messages indicate errors found during the editing of test output.
TESTRAN Interpreter Messages	TESTRAN Listing (TESTRAN editor SYSPRINT data set)	<p>*** IEGInn text</p> <p>IEG = TESTRAN message code</p> <p>Inn = Message serial number indicating the TESTRAN interpreter</p> <p>text = Message text</p>	Messages indicate errors found by the TESTRAN interpreter during execution of the program being tested.
TESTRAN Macro-Expansion Messages	Assembly Listing (Assembler SYSPRINT data set)	<p>ss,*** IEGMnn text</p> <p>ss = Severity code, which is one of the following:</p> <ul style="list-style-type: none"> <li>* Informational message; no effect on execution</li> <li>4 Warning message; successful execution is probable</li> <li>8 Error; execution may fail</li> <li>12 Serious error; successful execution is improbable</li> </ul> <p>IEG = TESTRAN message code</p> <p>Mnn = Message serial number indicating macro-expansion</p> <p>text = Message text</p>	Messages indicate errors in the position and syntax of TESTRAN statements. The assembler finds these errors when it expands TESTRAN statements (macro-instructions) into sequences of assembler language statements. If errors in a source statement cause errors in its expansion, the assembler may issue additional messages when it assembles the statements in the expansion. The additional messages do not have TESTRAN message codes and are not included in this appendix.

TESTSTRAN EDITOR MESSAGES

IEGE02 UNKNOWN MACRO

Explanation: During TESTSTRAN editing, an input record could not be related to a TESTSTRAN statement (macro-instruction) associated with the task that produced the data set.

System Action: The count of invalid records was incremented, and the record was ignored.

IEGE03 EXCESSIVE CHANGE DUMPS

Explanation: During TESTSTRAN editing, the output from an excessive number of DUMP CHANGES statements was selected for editing by the TESTSTRAN editor.

System Action: Only the output from the allowable number of DUMP CHANGES statements was edited.

User Response: To edit the output from subsequent DUMP CHANGES statements, repeat the job step without selecting the output from the DUMP CHANGES statements already edited.

IEGE04 INVALID RECORD--IGNORED

Explanation: During TESTSTRAN editing, an invalid or unreadable input record was encountered.

System Action: The count of invalid records was incremented, and the record was ignored.

IEGE05 EXCESSIVE INVALID RECORDS--EDIT DISCONTINUED

Explanation: During TESTSTRAN editing, the number of invalid or unreadable records in the data set exceeded the allowable limit.

System Action: The job step was terminated.

User Response: Determine whether the correct data set was used as input. If it was, recreate the data set by executing the problem program again.

IEGE06 EXCESSIVE OUTPUT

Explanation: During TESTSTRAN editing, the amount of edited output exceeded the limit specified in the PARM parameter of the EXEC statement for the job step being tested.

System Action: The job step was terminated.

User Response: Execute the job step again, specifying either a higher page limit or fewer output class identification numbers.

IEGE07 END OF TESTSTRAN EDIT--xxx STATEMENTS PROCESSED

Explanation: TESTSTRAN editing was completed.

In the message text, xxx is the number of TESTSTRAN statements executed by the TESTSTRAN interpreter.

IEGE08 INVALID OVERLAY RECORD

Explanation: During TESTSTRAN editing, an input record specified a change in an unknown overlay segment.

System Action: The record was ignored.

IEGE09 INVALID RELOCATION RECORD--EDIT DISCONTINUED

Explanation: During TESTSTRAN editing, an input record contained control section relocation information that did not correspond to the control section definitions of the program that was being tested.

System Action: The job step was terminated.

User Response: Determine whether the correct data set was used as input. If it was not, recreate the data set by executing the problem program again.

IEGE10 EXCESSIVE SECTION DEFINITIONS--ENTRY xxx

Explanation: During TESTSTRAN editing, the number of definitions of control, dummy, and blank common sections exceeded the limit allowed in the tested program.

In the message text, xxx is the entry name of the excess section.

System Action: Dumps and traces of the excess sections were printed in 4-byte hexadecimal format, except where this format was overridden by DATAM operands.

User Response: Reduce control sections, dummy sections, and blank common sections to the allowable number. Count each TESTRAN control section once for each time it is opened.

IEGE11 EXCESSIVE 'TEST AT'S

Explanation: During TESTRAN editing, the number of supervisor call (SVC) instructions inserted by TEST AT statements exceeded the limit.

System Action: Data resulting from the excess supervisor call instructions was ignored.

User Response: Reduce problem program addresses specified by TEST AT statements to the allowable number. Count each address once for each opening of the TESTRAN control section in which the address is specified.

IEGE12 EXCESSIVE 'TEST OPEN'S

Explanation: During TESTRAN editing, the opening of TESTRAN control sections by TEST OPEN statements exceeded the limit.

System Action: Data resulting from the excess control section openings was ignored.

User Response: Reduce TESTRAN control section openings to the allowable number.

IEGE13 UNABLE TO OPEN

Explanation: During TESTRAN editing, a required data set could not be opened because no DD statement was supplied for the data set.

System Action: The job step was terminated.

User Response: Supply the missing DD statement and execute the job step again.

IEGE14 IO ERROR

Explanation: During TESTRAN editing, an uncorrectable input/output error occurred.

System Action: The job step was terminated.

User Response: If the input/output error persists, have the computing system checked.

TESTRAN INTERPRETER MESSAGES

IEGI00 INVALID ADDRESS--IGNORED

Explanation: During execution of the TESTRAN interpreter, a TESTRAN statement referred to an address higher than the highest address in main storage.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, make sure that all address operands were specified correctly and were not modified. Also, check the contents of any registers referred to in the statement. Correct the error.

IEGI01 INVALID 'GO TO' AT xxx

Explanation: During execution of the TESTRAN interpreter, a GO TO or GO IN statement did not specify in its second operand the address of a TESTRAN statement in an open control section.

In the message text, xxx is the address in hexadecimal of the GO TO or GO IN statement.

System Action: The statement was ignored. The next sequential statement was executed.

User Response: If the job step is to be executed again, make sure that the second operand specified the address (symbolic name) of a TESTRAN statement and was not incorrectly modified. Also make sure that the control section containing the address will be open when the GO TO or GO IN statement is executed.

IEGI02 INACTIVE 'GO TO' AT xxx

Explanation: During execution of the TESTRAN interpreter, a GO TO or GO IN statement in an overlay program specified as its second operand and the address of a TESTRAN statement. This statement was in a control section that was not currently in main storage.

In the message text, xxx is the address in hexadecimal of the GO TO or GO IN statement.

System Action: The GO TO or GO IN statement was ignored. The next sequential statement was executed.

User Response: If the job step is to be executed again, make sure that the control section containing the specified address will be in main storage when the GO TO or GO IN statement is executed.

IEGI03 INVALID 'GO OUT' AT xxx

Explanation: During execution of the TESTRAN interpreter, a GO OUT statement was to be executed, but the associated GO IN statement had not saved a return address.

In the message text, xxx is the address in hexadecimal of the GO OUT statement.

System Action: The GO OUT statement was treated as a GO BACK statement in which the second operand was omitted.

User Response: If the job step is to be executed again, determine why the return address was missing, making sure that no attempt was made to save more than three return addresses.

IEGI04 NULL 'TEST OPEN' ENTRY POINT--ABEND

Explanation: During execution of the TESTRAN interpreter, a TEST OPEN statement did not specify as its second operand an entry point address in the problem program to which control could be returned.

System Action: The task was terminated abnormally.

User Response: Specify the entry point address in the TEST OPEN statement, making sure that the statement was not incorrectly modified.

Alternatively, avoid execution of this statement by listing it in the OPTEST operand of another TEST OPEN statement.

IEGI05 INVALID 'TEST AT'--IGNORED

Explanation: During execution of the TESTRAN interpreter, the second operand (address sublist) of a TEST AT statement specified an address that was outside the boundaries of the main storage assigned to the current task.

System Action: A supervisor call (SVC) instruction was not inserted at the erroneous address. Supervisor call instructions were inserted at valid addresses specified in the same sublist.

User Response: If the job step is to be executed again, make sure that the address was specified correctly and was not incorrectly modified. Correct the error.

IEGI06 EXCESSIVE OUTPUT REQUESTED

Explanation: During execution of the TESTRAN interpreter, the MAXP operand of a TEST OPEN statement specified a limit higher than the installation's limit on TESTRAN output.

System Action: The installation's limit was used instead of the limit specified by the statement.

User Response: If the job step is to be executed again, eliminate the MAXP operand, or specify a limit less than or equal to the installation's limit.

IEGI07 EXCESSIVE PROCESSING REQUESTED

Explanation: During execution of the TESTRAN interpreter, the MAXE operand of a TEST OPEN statement specified a limit higher than the installation's limit on processing by the TESTRAN interpreter.

System Action: The installation's limit was used instead of the limit specified by the statement.

User Response: If the job step is to be executed again, eliminate the MAXE operand, or specify a limit less than or equal to the installation's limit.

IEGI08 LIMIT OF ONE 'TEST OPEN' IN OVERLAY

Explanation: During execution of the TESTSTRAN interpreter, a second TEST OPEN statement was executed in an overlay program.

System Action: No control sections were opened on execution of the second TEST OPEN statement. Control was returned to the problem program at the address specified by the second operand.

User Response: If the job step is to be executed again, remove the second TEST OPEN statement from the overlay program. The one TEST OPEN statement allowed must be in the root segment. Its OPTEST operand should specify the names of other TEST OPEN statements for which control sections are to be opened.

IEGI09 'AT' LOCATION CONTAINS INVALID  
TESTSTRAN SVC

Explanation: During execution of the TESTSTRAN interpreter, a supervisor call (SVC) instruction was not inserted in the program being tested when the TESTSTRAN control section was opened by a TEST OPEN statement. The address in the program at which the supervisor call instruction should have been inserted was specified in a TEST AT statement. The supervisor call instruction would have called the TESTSTRAN interpreter.

System Action: The address in the TEST AT statement was ignored and a supervisor call instruction was not inserted.

User Response: If the job step is to be executed again, make sure that the address specified in the TEST AT statement (1) was correct, (2) was not incorrectly modified, and (3) was the address of an executable problem program instruction.

IEGI10 DUMP TRUNCATED AT END OF STORAGE

Explanation: During execution of the TESTSTRAN interpreter, a DUMP DATA or DUMP CHANGES statement specified an ending address that was higher than the highest address in main storage.

System Action: Only the storage from the starting address to the end of storage was dumped.

User Response: If the job step is to be executed again, make sure that the third positional operand specifies an address within storage and that it was not incorrectly modified.

IEGI11 'TEST OPEN' LIMIT REACHED

Explanation: During execution of the TESTSTRAN interpreter, TESTSTRAN control sections had been opened 255 times and another request to open a TESTSTRAN control section was found in the same task. TESTSTRAN control sections can be opened only 255 times during execution of one task.

System Action: No additional control sections were opened. Control was returned to the problem program address specified by the TEST OPEN statement that was executed most recently.

User Response: If the job step is to be executed again, count the number of times TESTSTRAN control sections are opened. A control section is counted once for each time it should be opened according to the logic of the program. Change the program to reduce the total openings if they exceed 255. If the total openings are fewer than 256, check for an uncontrolled loop that might cause repeated opening and closing of one or more control sections.

IEGI12 DUMP TRUNCATED AT TASK BOUNDARY

Explanation: During execution of the TESTSTRAN interpreter, a DUMP DATA or DUMP CHANGES statement specified an ending address that was outside the boundaries of the main storage assigned to the task.

System Action: Only the storage from the starting address to the task boundary was dumped.

User Response: If the job step is to be executed again, make sure that the second and third positional operands of the statement were specified correctly and were

not incorrectly modified. If the program is scatter loaded, both operands should specify addresses in the same control section.

IEGI13 INVALID 'SET VARIABLE' 'TO' ADDRESS

Explanation: During execution of the TESTSTRAN interpreter, a SET VARIABLE statement specified a variable at an address that was outside the main storage assigned to the task.

System Action: The SET VARIABLE statement was ignored.

User Response: If the job step is to be executed again, make sure that the address of the variable was specified correctly and was not incorrectly modified. Also check the contents of any registers referred to in the statement.

IEGI15 UNDEFINED COUNTER

Explanation: During execution of the TESTSTRAN interpreter, a SET COUNTER or TEST ON statement referred to a TESTSTRAN counter that was not in an open TESTSTRAN control section.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, define the counter with a TEST DEFINE statement in a control section that will be open when the counter is referred to.

IEGI16 TESTSTRAN CSECT ALTERED

Explanation: During execution of the TESTSTRAN interpreter, a control section containing TESTSTRAN statements was modified.

System Action: The task was terminated abnormally.

User Response: Find the error that caused the TESTSTRAN control section to be modified, correct it, and execute the job step again.

IEGI17 MAXIMUM PAGES EXCEEDED

Explanation: During execution of the TESTSTRAN interpreter, the limit on TESTSTRAN output was exceeded.

System Action: The task was terminated abnormally.

User Response: If excessive output was produced, check for errors in the statements that cause output and in the sequence in which they were executed. If the output was not excessive, specify a higher limit in the MAXP operand of the first TEST OPEN statement executed in the task. Then execute the job step again.

IEGI18 MAXIMUM STATEMENTS EXCEEDED

Explanation: During execution of the TESTSTRAN interpreter, the number of TESTSTRAN statements that can be processed during a single task exceeded the limit.

System Action: The task was terminated abnormally.

User Response: Check the test output for logical errors that would cause excessive processing. If no errors are found, specify a higher limit in the MAXE operand of the first TEST OPEN statement executed in the task. Then execute the job step again.

IEGI19 INVALID TESTSTRAN SVC--IGNORED

Explanation: Control was given to the TESTSTRAN interpreter by a supervisor call (SVC) instruction. The supervisor call instruction was not inserted by the TESTSTRAN interpreter in the current task.

System Action: No testing was performed. Control was returned to the location following the invalid supervisor call instruction.

User Response: If the job step is to be executed again, remove the invalid instruction or correct it.

IEGI20 INACTIVE TESTSTRAN SVC--IGNORED

Explanation: Control was given to the TESTSTRAN interpreter by a supervisor call (SVC) instruction that had been inserted during opening of a TESTSTRAN control section in another overlay segment. The segment containing the control section had been overlaid.

System Action: No testing was performed. The displaced problem program instruction was executed, and control was returned to the next sequential instruction.

User Response: If the job step is to be executed again, check all TEST AT statements to ensure that they specify problem program addresses in the same overlay segment. Correct any erroneous addresses.

IEGI21 INVALID 'TEST ON' BRANCH ADDRESS

Explanation: During execution of the TESTSTRAN interpreter, a TEST ON statement should have branched to another TESTSTRAN statement. The other statement was not in an open control section.

System Action: No branch occurred. The next sequential statement was executed.

User Response: If the job step is to be executed again, check the branch address which is specified by the fifth operand of the TEST ON statement. Ensure that the control section containing the address will be open when the TEST ON statement is executed.

IEGI22 INACTIVE 'TEST ON' BRANCH ADDRESS

Explanation: During execution of the TESTSTRAN interpreter, a TEST ON statement should have branched to another TESTSTRAN statement. The other statement was in an overlay segment not currently in main storage.

System Action: No branch occurred. The next sequential statement was executed.

User Response: If the job step is to be executed again, check the branch address, which is specified by the fifth operand of the TEST ON statement. Ensure that the control section containing the address will be in main storage when the TEST ON statement is executed.

IEGI23 'DUMP' TRUNCATED AT 65K BYTES

Explanation: During execution of the TESTSTRAN interpreter, a DUMP DATA or DUMP CHANGES statement specified dumping of a storage area containing more than 65,535 bytes.

System Action: Only the first 65,535 bytes of the specified area were dumped.

User Response: If the job step is to be executed again, check the starting and ending addresses for the dump; these are specified by the second and third positional operands. Ensure that the difference between the addresses will not exceed 65,535 bytes when the program is loaded. If the program is scatter loaded, both addresses must be in the same control section.

IEGI24 INACTIVE COUNTER

Explanation: During execution of the TESTSTRAN interpreter, a SET COUNTER or TEST ON statement referred to a TESTSTRAN counter in an overlay segment not currently in storage.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, define the counter with a TEST DEFINE statement that will be in storage when the counter is referred to.

IEGI25 INVALID DATA LENGTH

Explanation: During execution of the TESTSTRAN interpreter, the second and fourth operands of a TEST WHEN statement specified the location of data in registers or main storage. Both the type and length attributes of this data were specified by a DATAM operand. The data length exceeded the limit for the data type.

System Action: The statement was ignored. The next sequential statement was executed.

User Response: If the job step is to be executed again, correct the DATAM operand by specifying a data length and type that are consistent.

IEGI26 INVALID 'DUMP' ADDRESS

Explanation: During execution of the TESTRAN interpreter, a DUMP DATA or DUMP CHANGES statement specified a starting or ending address that was higher than the highest address in main storage.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, make sure that the second or third operand of the DUMP DATA or DUMP CHANGES statement was specified correctly and was not incorrectly modified. Also check the contents of any registers referred to in the operand.

IEGI27 INVALID 'WHEN' BRANCH ADDRESS

Explanation: During execution of the TESTRAN interpreter, a TEST WHEN statement should have branched to another TESTRAN statement. The other statement was not in an open control section.

System Action: No branch occurred. The next sequential statement was executed.

User Response: If the job step is to be executed again, check this branch address, which is specified by the last positional operand of the TEST WHEN statement. Ensure that the control section containing the address will be open when the TEST WHEN statement is executed.

IEGI28 INACTIVE 'WHEN' BRANCH ADDRESS

Explanation: During execution of the TESTRAN interpreter, a TEST WHEN statement should have branched to another TESTRAN statement. The other statement was in an overlay segment not currently in storage.

System Action: No branch occurred. The next sequential statement was executed.

User Response: If the job step is to be executed again, check the branch address, which is specified by the last positional operand of the TEST WHEN statement. Ensure that the control section containing the address will be in main storage when the TEST WHEN statement is executed.

IEGI29 INVALID SIGN ON DECIMAL FIELD

Explanation: During execution of the TESTRAN interpreter, the second or fourth positional operand of a TEST WHEN statement specified the address of a decimal value. The sign position of the decimal value contained an invalid bit configuration.

System Action: The TEST WHEN statement was ignored. The next sequential statement was executed.

User Response: If the job step is to be executed again, correct the sign in the rightmost byte of the decimal value.

IEGI30 ADDR1 GREATER THAN ADDR2

Explanation: During execution of the TESTRAN interpreter, a DUMP DATA, DUMP CHANGES, TRACE REFER, TRACE FLOW, or TRACE CALL statement specified a starting address that was higher than the ending address for the dump or trace.

System Action: The dump or trace was restricted to the single byte at the starting address.

User Response: If the job step is to be executed again, make sure that the second or third operand was specified correctly and was not incorrectly modified. Also check the contents of any registers referred to in the operand. If the program is scatter loaded, both operands should specify addresses in the same control section.

IEGI31 TRACE TABLE FULL AT xxx

Explanation: During execution of the TESTRAN interpreter, a TRACE CALL, TRACE FLOW, or TRACE REFER statement was executed when ten traces were already active.

In the message text, xxx is the address in hexadecimal of the statement.

System Action: A new trace was started, as specified by the statement. However, the tenth trace, the one that had been most recently started, was suspended.

User Response: If the job step is to be executed again, change the testing logic so that no more than ten traces are active at one time.

IEGI32 DEB UNAVAILABLE

Explanation: During execution of the TESTRAN interpreter, the second operand of a DUMP TABLE statement specified dumping of a data extent block (DEB). The associated data control block (DCB), specified by the third operand, was not currently open.

System Action: The DUMP TABLE statement was ignored.

User Response: If the job step is to be executed again, make sure that the data control block will be open when the DUMP TABLE statement is executed.

IEGI33 ILLEGAL 'TEST AT' DELETED FROM--xxx

Explanation: During execution of the TESTRAN interpreter, control was to be returned to the problem program at an address specified by a TEST AT statement. At the return address was a TESTRAN supervisor call (SVC) instruction that displaced either another supervisor call instruction or a privileged instruction. Before control was returned, the original instruction was replaced in the problem program.

In the message text, xxx is the return address in hexadecimal in the problem program.

System Action: If the original instruction was a privileged instruction, its execution caused abnormal termination of the task.

If it was a supervisor call instruction, it was executed normally and remained in the problem program until the TESTRAN interpreter received control from a supervisor call instruction inserted at some other address. Then, the original supervisor call instruction was again displaced by a TESTRAN supervisor call instruction.

User Response: If the original instruction was privileged, change the TEST AT statement so that it inserts the supervisor call instruction at another address. Then execute the job step again.

If the original instruction was a supervisor call instruction and if the job step is to be executed again, allow for the temporary displacement of the TESTRAN supervisor call instruction, or rewrite the TEST AT statement.

IEGI34 PROGRAM CHECK DURING 'GO BACK' --  
INSTRUCTION AT xxx

Explanation: During execution of the TESTRAN interpreter, control was to be returned to the problem program after execution of an instruction that was displaced by insertion of a TESTRAN supervisor call (SVC) instruction. Execution of the displaced instruction caused a program interruption.

In the message text, xxx is the address in hexadecimal of the TESTRAN supervisor call instruction.

System Action: The standard system exit routine, or the routine specified by a SPIE macro-instruction, was given control.

User Response: Correct the instruction causing the program interruption and execute the job step again.

IEGI39 INACTIVE FLAG

Explanation: During execution of the TESTRAN interpreter, a SET FLAG or TEST WHEN statement referred to a TESTRAN flag contained in an overlay segment not currently in main storage.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, define the flag with a TEST DEFINE statement that will be in storage when the flag is referred to.

IEGI41 UNDEFINED FLAG

Explanation: During execution of the TESTSTRAN interpreter, a SET FLAG or TEST WHEN statement referred to a TESTSTRAN flag not contained in an open TESTSTRAN control section.

System Action: The statement was ignored.

User Response: If the job step is to be executed again, define the flag with a TEST DEFINE statement in a control section that will be open when the flag is referred to.

IEGI42 INVALID 'TRACE STOP' ENTRY AT xxx

Explanation: During execution of the TESTSTRAN interpreter, the second operand of a TRACE STOP statement specified an address or sublist of addresses. One of these addresses was not the address of a TRACE statement and was, therefore, invalid.

In the message text, xxx is the invalid address in hexadecimal.

System Action: The invalid address was ignored. If the operand was a sublist, all traces corresponding to valid addresses were stopped.

User Response: If the job step is to be executed again, correct the invalid address.

IEGI43 'TRACE' STOPPED BY OVERLAY AT xxx

Explanation: During execution of the TESTSTRAN interpreter, the problem program loaded an overlay segment that overlaid all the TRACE statements for active traces.

In the message text, xxx is the address in hexadecimal of the instruction that caused the loading.

System Action: All traces were stopped. They were not automatically restarted when the segment containing the TRACE statements was reloaded.

User Response: If the job step is to be executed again, change the program so that the TRACE statements are not overlaid or be prepared to restart any traces that will be overlaid but will be required subsequently.

IEGI45 PROGRAM CHECK DURING 'TRACE' -- INSTRUCTION AT xxx

Explanation: During execution of the TESTSTRAN interpreter, a program interruption occurred during a trace of the problem program.

In the message text, xxx is the address in hexadecimal of the instruction that caused the interruption.

System Action: The standard system exit routine, or the routine specified by a SPIE macro-instruction, was given control. Active traces were not suspended.

User Response: If the job step is to be executed again, correct the instruction causing the program interruption.

IEGI47 'TRACE' STOPPED BY SVC AT xxx

Explanation: During execution of the TESTSTRAN interpreter, a LINK, XCTL, or RETURN macro-instruction was executed during a trace of the problem program.

In the message text, xxx is the address in hexadecimal of the supervisor call (SVC) instruction in the macro-expansion.

System Action: All traces were stopped. They were not automatically restarted when control was returned to the problem program.

User Response: If the job step is to be executed again, restart any traces that were stopped, but are required, upon return to the problem program.

IEGI48 FLOATING POINT REGISTER SELECTED  
NO FLOATING POINT HARDWARE  
JOB ABORTED

Explanation: During execution of the TESTSTRAN interpreter, a TESTSTRAN statement referred to a floating point register, but the computing system did not include the floating point option.

System Action: The task was terminated abnormally.

User Response: Either 'remove all references to floating point registers, and execute the job step again, or execute the job step on a computing system with the floating point option.

IEGM04 THIS MACRO ESTABLISHES CSECT xxx

Explanation: A TEST OPEN statement, named xxx, initiates assembly of a control section with the same name. This control section will contain all subsequent TESTSTRAN statements until the next TEST OPEN macro-instruction initiates a new control section.

System Action: The statement was processed normally. Severity code = \*.

#### TESTSTRAN MACRO-EXPANSION MESSAGES

IEGM01 TEST HAS NOT BEEN OPENED

Explanation: A TESTSTRAN statement precedes the first valid TEST OPEN statement.

System Action: The statement was deleted. Severity code = 8.

User Response: Precede the statement with a valid TEST OPEN statement.

IEGM05 xxx NOT A VALID OPERAND FOR yyy

Explanation: The first operand of a TEST statement is xxx. This operand is not valid following the operation field yyy.

System Action: The statement was deleted. Severity code = 8.

User Response: Correct the first operand.

IEGM02 NAME NOT SPECIFIED

Explanation: A TEST OPEN statement does not contain a symbol in its name field.

System Action: The statement was deleted. Severity code = 12.

User Response: Provide the required symbolic name.

IEGM06 xxx yyy ADDRESS. NOT SPECIFIED

Explanation: A required address operand was omitted from a TESTSTRAN statement whose operation field is xxx and whose first operand is yyy.

System Action: The statement was deleted. Severity code = 8.

User Response: Provide the required address operand.

IEGM03 ENTRY POINT NOT SPECIFIED

Explanation: The second positional operand (problem program entry point) was omitted from a TEST OPEN statement.

System Action: The statement was processed normally. Severity code = \*.

User Response: No response is required if the TEST OPEN statement never receives control directly, but instead is referred to by the OPTTEST operand of another TEST OPEN statement. If the TEST OPEN statement does receive control directly, the omitted operand should be supplied.

IEGM07 THIS TEST DEFINE xxx HAS NO xxxS

Explanation: The third positional operand (flag or counter sublist) was omitted from a TEST DEFINE statement. The second positional operand, xxx, is either COUNTER or FLAG.

System Action: The statement was deleted. Severity code = 8.

User Response: Provide the required sublist of flag or counter names.

IEGM08 xxx NOT A VALID TEST DEFINE OPERAND

Explanation: The second positional operand of a TEST DEFINE statement is xxx. This operand is invalid.

System Action: The statement was deleted. Severity code = 8.

User Response: Correct the second operand. It must be either COUNTER or FLAG.

IEGM09 MACRO NUMBER xxx IN yyy

Explanation: An identification number, xxx, was assigned to a TESTRAN statement. This statement is in a control section named yyy, which is the name of the preceding TEST OPEN statement.

System Action: The statement was processed normally. Severity code = \*.

User Response: Keep the assembler source and object program listing for comparison with the listing of TESTRAN edited output. The statement identification number, which appears in both listings, identifies all output produced by the statement.

IEGM10 SELECT CODE INVALID AND IGNORED

Explanation: The SELECT operand of a TESTRAN statement does not specify a valid TESTRAN output class.

System Action: The statement was processed, but the invalid operand was ignored. Severity code = 4.

User Response: Specify a valid output class number (an integer from 1 to 8), or compensate for the error by changing the PARM parameter of the EXEC statement for the TESTRAN editor.

IEGM12 xxx NOT A VALID OPERATOR

Explanation: The third positional operand of a TEST WHEN statement is xxx. This operand is not a valid logical or relational operator.

System Action: The statement was deleted. Severity code = 8.

User Response: Specify a valid logical operator (AND or OR) or relational operator (LT, LE, EQ, NE, GT, or GE).

IEGM13 INVALID LITERAL TYPE CODE

Explanation: An operand of a TESTRAN statement is a literal in which the type code is either absent or invalid.

System Action: The statement was deleted. Severity code = 8.

User Response: Correct the operand by specifying a valid type code following the equal sign (=) of the literal.

IEGM14 BOTH xxx AND yyy CANNOT BE LITERALS

Explanation: The second and fourth positional operands of a TEST WHEN statement are xxx and yyy, respectively. Both are literals. Because the arithmetic relationship between two literals is constant, a test of this relationship would be meaningless.

System Action: The statement was deleted. Severity code = 8.

User Response: Replace one literal with pointer to a main storage location, a register, or a TESTRAN counter.

IEGM17 DATAM IGNORED ON THIS FORM OF TEST WHEN

Explanation: A DATAM operand appears in a TEST WHEN statement that tests the condition of a TESTRAN flag, or a relationship between TESTRAN flags. The operand is invalid in this context.

System Action: The statement was processed, but the invalid operand was ignored. Severity code = 4.

User Response: Omit the DATAM operand, or rewrite the statement to test a relationship between arithmetic variables.

IEGM18 FORMAT UNKNOWN. 1 BYTE HEX ASSUMED

Explanation: In a SET VARIABLE or TEST WHEN statement, two operands specify the location of data, which is in registers or main storage. The attributes of this data are not defined in the symbol table nor are they specified by a DATAM operand. The data is, therefore, assumed to be hexadecimal with a length of one byte.

- System Action: The statement was processed normally. Severity code = \*.
- User Response: If a 1-byte hexadecimal format is not intended, provide a DATAM operand that specifies the correct attributes.
- IEGM19 TEST WHEN WRITTEN IMPROPERLY
- Explanation: The format of a TEST WHEN statement is invalid.
- System Action: The statement was deleted. Severity code = 8.
- User Response: Correct the error in the format.
- IEGM20 NO RIGHT PAREN IN OPERAND xxx
- Explanation: A positional operand of a TESTRAN statement is an explicit or indexed implied address. In this operand, the right parenthesis was omitted. The position of the operand in the operand field is xxx.
- System Action: The statement was processed; the right parenthesis was assumed to be present. Severity code = 4.
- User Response: Check the source and object program listing to determine if assumption of the parenthesis resulted in correct processing of the statement. Rewrite the operand if the processing was not correct.
- IEGM31 COMMENT IS INVALID
- Explanation: In a DUMP COMMENT statement, the second positional operand (a programmer-written comment) either was omitted or is invalid. If invalid, the operand either is shorter than three characters (including delimiting apostrophes), or does not contain one or both of the required apostrophes.
- System Action: The statement was deleted. Severity code = 8.
- User Response: Specify or correct the comment operand.
- IEGM32 xxx NOT A VALID TABLE TYPE
- Explanation: The second positional operand of a DUMP TABLE statement is xxx. This operand is invalid.
- System Action: The statement was deleted. Severity code = 8.
- User Response: Correct the second operand. It must be DCB, DEB, or TCB.
- IEGM33 INVALID REGISTER NOTATION
- Explanation: The second positional operand (a register sublist) of a DUMP PANEL statement contains invalid register notation.
- System Action: The statement was processed; the invalid operand was ignored and dumping of all registers was assumed. Severity code = 4.
- User Response: No response is necessary.
- IEGM34 INVALID TYPE CODE IN xxx
- Explanation: The operand DATAM=xxx contains an invalid type code.
- System Action: The statement was processed, but the invalid operand was ignored. Severity code = 4.
- User Response: Correct the DATAM operand.
- IEGM40 A REQUIRED ADDRESS NOT SPECIFIED
- Explanation: This message occurred for either of two reasons:
- The second positional operand (the starting address for a trace) was omitted from a TRACE CALL, TRACE FLOW, or TRACE REFER statement.
  - The third positional operand (the ending address for a trace) was omitted from a TRACE CALL statement.
- System Action: The statement was deleted. Severity code = 8.
- User Response: Provide the required address operand.

IEGM41 THIS TRACE STOP STOPS ALL TRACES

Explanation: The optional second positional operand (trace sublist) was omitted from a TRACE STOP statement. This statement will, therefore, stop all active traces.

System Action: The statement was processed normally. Severity code = \*.

User Response: If all traces should not be stopped, provide the optional trace sublist operand to specify only those traces that are to be stopped.

IEGM42 COMMENT IS INVALID AND IGNORED

Explanation: The COMMENT operand of a TRACE statement is invalid. The operand either is shorter than three characters (including delimiting apostrophes), or does not contain one or both of the required apostrophes.

System Action: The statement was processed, but the invalid operand was ignored. Severity code = 4.

User Response: Correct the COMMENT operand.

IEGM50 2ND AND 3RD OPERANDS MUST BE PRESENT

Explanation: One or more required positional operands were omitted from a SET statement.

System Action: The statement was deleted. Severity code = 8.

User Response: Provide the required operand or operands.

IEGM51 SET FLAG CONDITION MUST BE =0 or =1

Explanation: The third positional operand (condition) of a SET FLAG statement is invalid.

System Action: The statement was deleted. Severity code = 8.

User Response: Write the third operand as =0 or =1, or as the symbolic name of a TESTRAN flag.

- Address
  - as an external reference 30
  - assembled as a constant 19,71
  - declared in a USING statement 19,30
  - explicit 30,71
  - indexed 17,27,64,71
  - test point 15,67
- Ampersand 29,71
- Apostrophe 29,71
- Area
  - (see storage area)
- ASM (job step) 35,39,42,45,73,74
- ASMEC (cataloged procedure)
  - definition 73
  - use 35
- Assembler
  - E-level assembler program 35,39,41,44
  - listing 9,49
  - options 39,42,45
  - processing of TESTRAN
    - macro-instructions 12
  - symbol tables 24
  - use with TESTRAN 11
- Assembly
  - job control statements for 35,39,41,44
  - listing 8,9,30
  - of address operands 19,71
  - of problem program and TESTRAN 11,30
- Assignment functions 13,70
- Asynchronous exit routines 27
- Attributes 23,64,65,69,70
- Base address
  - for addressing dummy control sections 19,28
  - for addressing other object modules 30
- Blank common control section 24
- Branch
  - by a TESTRAN statement 12,66,68,69
  - tracing of 26,64
- Branching functions 13,69
- Call library 37,41,44,48
- CALL macro-instruction 25,26,32,64
- Cataloged procedures (IBM-supplied)
  - definitions 73,74
  - use 35,36,38,39,41,44
- Chained opening
  - definition 67
  - examples 32,33
- Change dump 17,64
- Changes
  - in index values 17
  - to a dummy control section 19
  - to a storage area 16,64
- Class (of test information)
  - definition by a SELECT operand 29,33,65,67
  - identification in a TESTRAN listing 53-61
  - selection for printing 38,46
- Class identification number
  - in a job control statement 38,46
  - in a SELECT operand 29,65,67
  - in a TESTRAN listing 53-61
- Class number
  - (see class identification number)
- Coding conventions 63
- Commas 63
- COMMENT operand
  - description 71
  - example 29
  - function 65
- Comments
  - in the comments field 12
  - in the operand field 29
- Common control section 24
- Completion
  - of testing 31
  - of a timer interval 28
- Condition
  - condition code 26,55,59
  - condition testing 11,68
  - error conditions 8
- Conditional branching 68
- Constants 12
- Control block 20-22,64,65
- Control dictionary
  - handling by the linkage editor 37,41,44,47
  - inclusion in a load module 12,36,40,43,46
  - production by the assembler 12
- Control flow, tracing of 25
- Control information
  - recorded by the TESTRAN interpreter 11
  - used by the TESTRAN editor 12
  - (see also symbol tables)
- Control sections
  - defined by TEST OPEN (see TESTRAN control section)
  - map of 20
  - replaced by the linkage editor 31,36,41,43,47
- Conventions
  - for coding TESTRAN statements 63
  - for describing TESTRAN statements 71
- Count
  - line count for assembly listing 36,39,42,45
  - page count for TESTRAN listing 39,47,67
- Counter
  - (see TESTRAN counter)
- Data control block 21,22,64,65
- Data extent block 21,22,64,65
- Data set
  - (see TESTRAN data set)
- Data types
  - printing formats 51
  - specification 71
- DATAM operand
  - description 71
  - examples 23
  - function 65,69,70
- DCB
  - macro-instruction 20,28
  - operand 20,21,65,71
  - (see also data control block)

DEB  
   operand 20,21,65,71  
   (see also data extent block)  
 Decision-making functions 13,68  
 Default  
   assembler options 39,40,42,44,45  
   printing format 24  
 Dictionary  
   (see control dictionary; external symbol  
   dictionary)  
 Displacement 30,71  
 DSECT operand  
   description 71  
   examples 18,19,28  
   function 65  
 Dummy control section  
   addressing of 28  
   describing another module 30  
   dumping changes to 19  
   dumping of 17  
   tracing references to 28  
 Dump  
   definition 11  
   examples 15,16,19-21,29  
   formats 53-56  
 DUMP statements  
   examples  
     DUMP CHANGES 16,23  
     DUMP COMMENT 29  
     DUMP DATA 15,18,23,30  
     DUMP MAP 20  
     DUMP PANEL 20-23  
     DUMP TABLE 20  
   formats 71  
   functions 64  
 Dynamic serial program 34  
  
 EDIT (job step) 38,45,74  
 Editing  
   linkage editing 11,12,36,39,41,44  
   TESTSTRAN editing 11,12,38,44  
 END statement 14-34  
 Entry point  
   in an END statement 14,15  
   in an ENTRY statement 31  
   in a TEST OPEN statement 15,66  
 Entry point register 66  
 ENTRY statement  
   assembler 33  
   linkage editor 31  
 Error  
   detected by the TESTSTRAN interpreter  
     49,56  
   diagnostic message 56,62,75-88  
 ETXR operand 28  
 Execution, job control statements for  
   34,41,44  
 Exit routine 27  
 Exponent 51  
 External reference 30,33  
 External symbol 30,64,69,70  
 External symbol dictionary 30  
 EXTRN statement 33  
  
 Field  
   (see operand field; storage field)  
 Flag  
   (see TESTSTRAN flag)  
  
 Format  
   printing format  
     control of 22,31  
     of data types 51  
     of a TESTSTRAN listing 49  
   statement format 71  
  
 GO  
   job step 42,45,47,74  
   load module 20,37,41,44,73,74  
 GO statements  
   example (GO TO) 17  
   formats 71  
   functions 69  
  
 Hexadecimal  
   as a default format 31,69  
   as an implied data type 23  
  
 Implicit control section 29,31  
   (see also TESTSTRAN control section)  
 Indexed addresses 19,27,64,71  
 Internal subroutine 69  
 Interpretive execution 26,27  
  
 Job control statements, writing of 35  
 Job library 37,42,45  
  
 Keyword operands 63  
  
 Length attribute  
   of a symbol 64  
   specified by a DATAM operand  
     23,65,69-71  
 Library  
   (see call library; job library;  
   procedure library)  
 Limits on traces 27  
 Linkage editing  
   job control statements for 36,39,41,44  
   of problem program with TESTSTRAN  
     11,31,33  
 Linkage functions 12,66  
 Listing  
   (see assembly listing; TESTSTRAN listing)  
 Literal 68-71  
 LKED  
   cataloged procedure 36,73  
   job step 36,39,42,45,73,74  
  
 MACRO ID  
   in an assembly listing (see MACRO  
   NUMBER)  
   in a TESTSTRAN listing 49-61  
 MACRO NUMBER 86  
 Macro-instruction  
   ATTACH 28,34  
   CALL 25,26,32,64  
   DCB 20,28  
   GET, PUTX 28  
   IDENTIFY, LINK, LOAD, XCTL 34  
   OPEN 20  
   RETURN 8,9  
   SAVE 8,9,15-20,25,28  
   SPIE, STIMER 28  
   TESTSTRAN 12  
   (see also TESTSTRAN statements)  
 MAXE operand 66,67

Maximum number  
   of dummy control section formats 65  
   of executed TESTRAN statements 67  
   of internal subroutine levels 69  
   of pages in a TESTRAN listing 39,47,67  
   of traces 27  
 MAXP operand 66,67  
 Messages 56,62,75  
  
 NAME operand  
   description 71  
   examples 23  
   function 65  
  
 Opening of a TESTRAN control section 66,67  
 Operand field 63  
 Operation code  
   of a dumped instruction 24,51  
   of a TESTRAN statement 12  
 OPTEST operand  
   description 71  
   examples 32,33  
   function 67  
 Options  
   assembler 35,39,42,45  
   linkage editor 36,40,42,45  
   TESTRAN editor 38,46  
   (see also task options)  
 Output identification  
   printing of 52  
   specification of 66  
 Overlay program 33  
  
 PARM parameter 35-46  
 Positional operands 63  
 Printing format  
   control of 22,31  
   of data types 51  
   of a TESTRAN listing 49  
 Procedure library 35  
 Program status word, dumping of 20  
 PSW  
   (see program status word)  
  
 Recording functions 12,14,64  
 Reference  
   external reference 30,33  
   reference between overlay segments 33  
   tracing of references 25  
 Registers  
   dumping of 20  
   specification in TESTRAN statements 71  
 RENT option 36,40,43,46  
 Reprocessing  
   of a load module 31,37,41,43,47  
   of a TESTRAN data set 29  
 Return of control 15,17,34,53,66,69  
 REUS option 36,40,43,46  
 Reusability 34  
   (see also RENT option; REUS option)  
  
 Scale attribute 23,52,69-71  
 Scatter loading 27  
 Segment 33,34  
 SELECT operand  
   description 71  
   examples 29,30,33  
   function 65,67  
  
 Selective retrieval  
   classification of information for 29,33  
   selection of classified information 38,46  
 SET statements 70,71  
 Speed  
   of the TESTRAN editor 38,46  
   of the TESTRAN interpreter 66  
 Storage area  
   allocated 22,24,27,54  
   defined by indexed addresses 17  
   described by a dummy control section 17,19  
   described by a symbol table 24  
   dumping changes to 16  
   dumping of 14  
   length of 64,65  
   specification of 64  
   tracing references to 25  
 Storage field  
   defined by a DATAM operand 23,31,65  
   defined by a DS or DC statement 24  
 Storage map, dumping of 20  
 Storage requirements  
   of the TESTRAN editor 38,46  
   of the TESTRAN interpreter 66  
 Sublist 63  
 Subroutine call  
   by a GO IN statement 69  
   tracing of 26,64  
 Supervisor call  
   supervisor call instruction 54,67  
   tracing of 26,64  
 SVC instruction  
   (see supervisor call instruction)  
 Symbol tables  
   handling by the linkage editor 12,31,36,37,40,41,43,44,46,47  
   production by the assembler 12,35,39,42,45  
   use by the TESTRAN editor 12,24  
 SYNAD operand 28  
 System output 49  
 System table  
   (see control block)  
  
 Task control block 21,22,65  
 Task options 66,71  
 TASME (cataloged procedure)  
   definition 73  
   use 39  
 TASMEG (cataloged procedure)  
   definition 74  
   use 41  
 TASMEGED (cataloged procedure)  
   definition 74  
   use 44  
 TCB  
   operand 20,21,65,71  
   (see also task control block)  
 Test information  
   classification for selective retrieval 29,33  
   recording and printing of 11  
   selective retrieval of 38,46  
 TEST option  
   assembler 35,39,42,45  
   linkage editor 36,40,42,46

Test point  
   definition of 15  
   identification of 49,52  
   specification of 67  
   SVC instruction at 54

TEST statements  
   examples  
     TEST AT 14,30,33  
     TEST CLOSE 34  
     TEST OPEN 15,29,32,33  
   formats 71  
   functions 66-68

TESTSTRAN control section  
   chained opening by OPTTEST 32,33,67  
   closing by TEST CLOSE 34,66  
   definition by TEST OPEN 66,85  
   insertion in overlay segments 33  
   replacement by the linkage editor  
     31,37,41,43,47

TESTSTRAN counter  
   definition of 66,67  
   setting of 70  
   testing of 68

TESTSTRAN data set  
   creation by the TESTSTRAN interpreter 11  
   definition by a SYSTEST DD statement  
     37,38,42,45,74  
   processing by the TESTSTRAN editor  
     29,38,46

TESTSTRAN editing, job control statements  
   for 38,44

TESTSTRAN editor  
   definition 11  
   listing  
     (see TESTSTRAN listing)  
   storage requirements 38,46

TESTSTRAN flag  
   definition of 66,67  
   setting of 70  
   testing of 68

TESTSTRAN interpreter  
   control of  
   definition 11  
   linkage to 12,15,66

TESTSTRAN listing  
   commenting the listing 29

  example of 9,49  
   interpretation of 49  
   maximum page count 39,47,67

TESTSTRAN macro-instructions 12  
   (see also TESTSTRAN statements)

TESTSTRAN messages 75

TESTSTRAN output  
   (see TESTSTRAN listing)

TESTSTRAN services  
   description 8  
   requests for 11,12

TESTSTRAN statement trace 49,62

TESTSTRAN statements  
   examples of 14-34  
   execution of 12,15  
   functions 12,63  
   output of 52-62  
   structure and format 12,71

Trace  
   definition 11  
   examples 26  
   printing formats 58-61  
   shifting a trace 27,28  
   starting a trace 25  
   stopping and restarting a trace  
     25-27,34

Trace area  
   identification of 59,60  
   limitation of 27  
   specification of 64

TRACE statements  
   examples 25-29  
   formats 71  
   functions 64

TTED (cataloged procedure)  
   definition 74  
   use 37

Type attribute  
   of a symbol 64  
   specified by a DATAM operand  
     23,65,69-71

USING statement 18,19,28

Variable 13,70

READER'S COMMENTS

Title: IBM System/360 Operating System  
TESTRAN

Form: C28-6648-0

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 \_\_\_\_\_ fold

Please check the items that describe your position:

___ Customer personnel	___ Operator	___ Sales Representative
___ IBM personnel	___ Programmer	___ Systems Engineer
___ Manager	___ Customer Engineer	___ Trainee
___ Systems Analyst	___ Instructor	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:

CUT ALONG LINE

fold

staple

stapl

fold

fol

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

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS  
 DEPT. D58


Printed in U.S.A.

C28-6648-0

fold



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

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

stapl

**IBM**

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

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