Time Sharing Operating System (TSOS)

Language Processor Programming Reference Manual

July 1971
DJ-008-2-00

Marketing Publications
Building 204-2
Cherry Hill, N.J.

RCA

First Printing: June 1971 (DJ-008-2-00)

The following revision to this manual has been issued:

July 1971 (DJ-008-2-01)

PREFACE

This manual describes the Assembler, the COBOL and FORTRAN background compilers, and the Post-Assembly and Post-Compilation Diagnostic Routines. It also discusses briefly other routines which are helpful to a programmer after execution of a language processor program.

The reader of this manual should be familiar with the Assembly, COBOL, and FORTRAN languages described in their respective reference manuals. Furthermore, the assumption is made that programmers using the routines described in this text are familiar with the fundamentals of TSOS.

This manual contains six sections and delineates the basic concepts necessary to enter, compile and test the logic of programs under TSOS.

Section 1: Cobol Background Compilation – This section describes the methods for COBOL source program retrieval, object module storage, source and object program listing, Internal Symbol Dictionary generation, and diagnostic file generation. Included, as a part of the descriptive material are several sample compilations. This section covers the use of both the RCA COBOL background compiler and the ANSI COBOL background compiler.

Section 2: FORTRAN Background Compilation – This section contains information relating to the processing of FORTRAN programs. Included are discussions of compiler-source language compatibility, object module generation, source input specification, listing generation, and Internal Symbol Dictionary and diagnostic file generation. In addition, a description of the FORTRAN object time facilities has been given at the close of the section.

Section 3: Assembly – This section defines the facilities provided in TSOS for the processing of Assembly language programs. This information includes source input retrieval, object module transcription and listing, Internal Symbol Dictionary and diagnostic file generation.

Section 4: Post Source Language Translation Routines – This section presents an overview of the Post-compilation Diagnostic Routines. A description has also been included on the use of the Interactive Debugging Aid as a post-compilation diagnostic tool. In addition, information on immediate, as well as delayed execution of, object programs has been given at the close of the section.

Section 5: Post-compilation Diagnostic Routine (BDIAG) – This section describes the TSOS interactive diagnostic routine designed to provide programmers operating from remote terminals with diagnostic information concerning background COBOL and/or FORTRAN compilations.

Section 6: Post-assembly Diagnostic Routine (ADIAG) – This section describes the TSOS interactive diagnostic routine designed to provide programmers operating from remote terminals with diagnostic information concerning assembler executions.

CONTENTS

Page

1. COBOL BACKGROUND COMPILATION

INTRODUCTION

The RCA and ANSI COBOL Background Compilers (BGCOB and ANSICOB) are nonconversational, Class II (pageable) programs which run under control of RCA time sharing systems. Both compilers are direct access oriented programs. The compilation facilities provided by the two COBOL compilers are:

1. Source input retrieved from a cataloged file, from a card deck, from a remote terminal, or from cataloged COBOL source-library file.

2. A generated object module can be written on the task's Object Module File and then punched on cards.

3. Requested listings written on temporary files, and automatically printed or written on permanent files for printing at the user's convenience.

4. An Internal Symbol Dictionary (ISD) to be used by the Interactive Debugging Aid (IDA) for debugging at program execution time.

5. A diagnostic file generated for query through the Post-Compilation Diagnostic Routine (BDIAG).

Before compilation, the COBOL Programmer at a remote terminal may invoke the interactive COBOL Syntax Checker Routine (for RCA COBOL only) or the COBOL Program Development Subsystem to detect and correct syntactical errors in source programs, and to obtain a cataloged indexed sequential file containing the corrected source program.

During compilation, the COBOL Background Compiler operates in the nonconversational mode; however, these can be dynamic communication between the compiler and the programmer in the sense that the programmer may type a COBOL program at a terminal or type a BASIS statement which directs the compiler to compile a program from a COBOL Source Library.

The programmer can request that the COBOL compiler produce a diagnostic file of the program's compilation errors. In that case, the programmer may, after compilation, invoke the Interactive Post-Compilation Diagnostic Routine to interrogate this file. In this way, he may obtain diagnostic information about his compilation, such as the number of errors noted by the compiler and a description of any such errors. Once in possession of this information, the programmer can decide whether to correct the source program and recompile it, or to execute the object program.

At object program execution time, the programmer at a remote terminal can employ the IDA (Interactive Debugging Aid) System to aid in debugging COBOL object programs. If an Internal Symbol Dictionary (ISD) was created at compilation time, the IDA user will be able to refer to data fields, entry points and instructions by their COBOL source data names, procedure names, and sequence numbers. Without an ISD, the programmer must use hexadecimal addresses to refer to them.

EQUIPMENT CONFIGURATION

The COBOL Background Compilers each require approximately 100,000 bytes of virtual memory. The following minimum TSOS configuration is required for COBOL compilation:

1–Processor (262KB)
2–Direct Access Units
1–Paging Drum

In addition, the system must include a Communications Controller Multichannel (CCM) with associated buffers and remote terminals, if it is to support conversational users.

Optional Equipment

Direct Access Storage (for the COBOL source library)
1–Card Reader (card input)
1–Card Punch (if the object module is to be punched as cards.)
1–Printer (for listings)

COMPILER FEATURES

Source Input

The COBOL Background Compiler reads source input by using the RDATA macro. RDATA instructs the Executive to read an input record from the SYSDTA file. The SYSDTA file may be equated to a cataloged file on direct-access devices, a card reader, or a remote terminal. (See Figure 1-1.)

Disc-Resident Source Files

A COBOL source file which resides on direct-access must be either a Sequential Access Method (SAM) file or an Indexed Sequential Access Method (ISAM) file. In the latter case, the 8-byte numeric keys must be situated in the first 8 data bytes of the record. ISAM files with this construction may be directly created, updated, and edited by the File Editor (see the File Editor Reference Manual) or EDT (see the EDITOR Reference Manual).

If the compiler is to retrieve its input from a direct access file, it is the programmer's responsibility to ensure that the source input is contained in a cataloged file and to use the Executive command SYSFILE to direct SYSDTA to his source file. (See examples at the end of this section.)

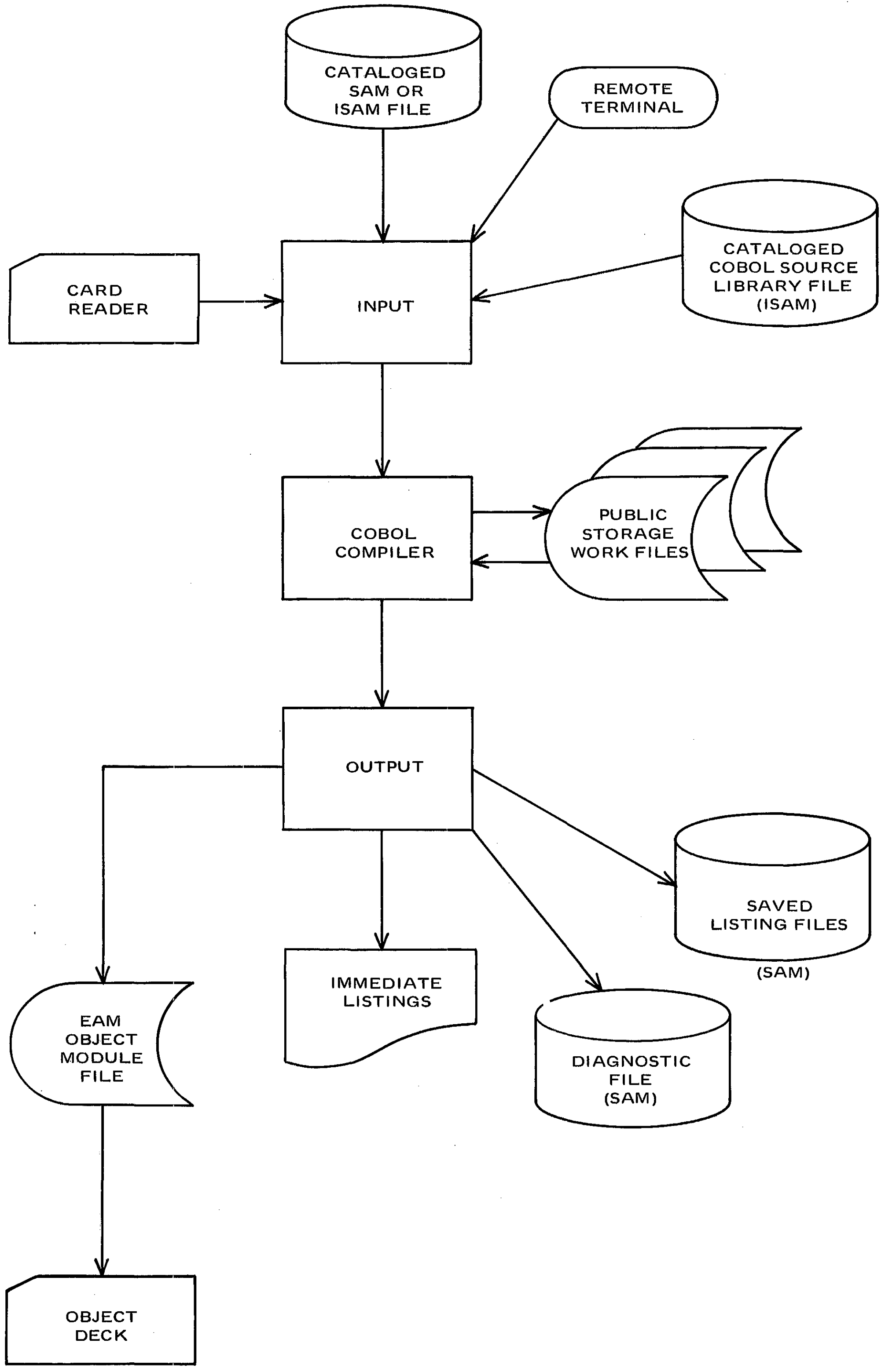


FIGURE 1-1. COBOL BACKGROUND COMPILER'S INPUT-OUTPUT FLOW CHART
(THE DIRECT ACCESS UNITS SHOWN ARE NOT NECESSARILY DIFFERENT)

Note: If the programmer wishes to do any additional processing after the end of the compilation, such as the Library Maintenance Routine, care must be taken at that time to redirect SYSDTA from the COBOL source file to whatever source is required by the subsequent routines. For instance, in a conversational task a programmer wishing to redirect SYSDTA to the terminal after compilation would issue the following command:

```
/SYSFILE SYSDTA=(SYSCMD)
```

Card Input

A programmer may wish to enter an entire compilation as a background task in the form of a card deck containing a /LOGON card, cards containing commands to the compiler, the COBOL source program, and a /LOGOFF card. The operator places the deck in the reader and initiates execution of the task by entering the RCARD command on the console, and the entire card deck is spooled in and then executed. In this case, the programmer should direct SYSDTA so that the compiler can take its input from the spooled-in file, thus:

```
/SYSFILE SYSDTA=(SYSCMD)
```

In a conversational task, SYSDTA is by default the remote terminal from which the task was initiated. A conversational programmer initiating a compilation from a terminal might wish to instruct the compiler to take its input from a source deck which the programmer had previously requested the operator to place in the reader. In this case, the programmer should redirect SYSDTA from the terminal to the card reader by giving the command:

```
/SYSFILE SYSDTA=(CARD)
```

The operator receives a message requesting confirmation that the deck is actually in the reader. SYSDTA should be redirected from the reader as soon as possible so as to free the card reader for other tasks.

Terminal Input

The conversational programmer may enter the source program directly from a remote terminal. In this case, the programmer must be sure that SYSDTA is directed to the terminal, redirecting it there, if necessary, by the command:

```
/SYSFILE SYSDTA=(SYSCMD)
```

before beginning a compilation.

The compiler types an asterisk (*) on the terminal, indicating that it is ready to receive source input. The programmer types a line of COBOL source, not longer than 80 characters and conforming to the usual COBOL margin requirements. The compiler reads this line, then types another asterisk (*) to signify that it is ready for another input line. This process goes on until the programmer has entered the whole source program and signified end of input by typing a slash followed by an asterisk (/*) instead of another input record. The compiler then takes control and proceeds to compile the source program. An input error will require that the programmer reload the compiler.

Use of SYSFILE Command

The Executive command SYSFILE allows the programmer to direct SYSDTA to a cataloged source file or to the system card reader, to redirect SYSDTA to its primary assignment, or to the command input stream.

```
SYSFILE      SYSDTA=      filename
                          (CARD)
                          (PRIMARY)
                          (SYSCMD)
```

filename

Identifies the name of the cataloged source file. The command is invalid if the file is not cataloged.

(CARD)

Identifies the system card reader.

(PRIMARY)

Identifies the primary assignment of SYSDTA.

(SYSCMD)

Identifies the command input stream.

Generated Object Modules

The compiler can generate an object module on public storage EAM space. The programmer may obtain punched cards by issuing a PUNCH * command.

Disc-Resident Object Modules

The compiler automatically writes the object module on an EAM file unless the programmer specifically inhibits this by means of the following command:

```
/PARAM DISC=NO
```

Because files created by EAM are temporary (at most, they remain available for the duration of the task in which they are created), the programmer who wishes to keep an object module in a permanent, direct access file should do so by invoking the Library Maintenance Routine (for a description of this routine, refer to the Utility Routines Reference Manual).

Object Module Card Decks

If the programmer wishes to receive an object module in the form of a deck of punched cards, he must be careful to issue a /PUNCH * command after the end of compilation and before logging off. This command instructs the system to punch out the EAM file; thus the programmer who wants an object deck must not have previously issued the command:

```
/PARAM DISC=NO
```

which would preclude the object module's being written on the task's Object Module File.

Furthermore, once the PUNCH command has been issued, the EAM object module file will no longer be available to the programmer. If he wishes to save the object module in a library, he should do so by using the Library Maintenance Routine before issuing the PUNCH command.

Class I and II Object Modules

The Compiler can produce either Class I or Class II object modules. Class II programs are pageable and use virtual memory. Class I programs use resident memory, are not pageable and are TOS/TDOS compatible.

The programmer should specify the type of object module required by using the /PARAM command:

/PARAM CLASS=1	for Class I object modules
/PARAM CLASS=2	for Class II object modules (default option)

Source and Object Listings

Immediate Listings

To obtain compiler listings, the programmer must specify the following parameters:

/PARAM LIST=YES	For source listing (NO is the default.)
/PARAM OBJLST=YES	For object listing (NO is the default.)
/PARAM XREF=YES	For cross-reference listing (NO is the default.)
/PARAM DIAG=YES	For diagnostic listing (YES is the default.)
/PARAM MAP=YES	For locator map listing (YES is the default.)

The compiler writes these listings on temporary files and then issues a spoolout PRNT macro causing them to be printed. The listings are spooled out as a separate task independent of the programmer's current task.

Saved Listings

If the programmer wishes to have any of these listings saved on a permanent file, he should specify whichever of the following parameters is applicable:

/PARAM SAVLST=SOURCE	For source listing
/PARAM SAVLST=OBJECT	For object listing
/PARAM SAVLST=LOCMAP	For maps, cross reference listing and diagnostic listing
/PARAM SAVLST=ALL	For all of the above

The specified listings are not printed when the programmer logs off after the compilation, but are available as permanent files so that the programmer may issue a /PRINT command whenever convenient. (For a discussion of the SPOOLOUT command, PRINT, refer to the Utility Routines Reference Manual.)

To print a SAVLST file, the programmer must use the /PRINT command with the edit option. For example:

```
/PRINT X1.SOURCE,SPACE=E
```

X1 is the program name and SOURCE identifies the listing required.

Thus X1.SOURCE is the fully qualified name of the file containing the source listing of program X1.

To request the printing of a saved object listing for program X2, the programmer would issue the following PRINT command:

```
/PRINT X2.OBJECT,SPACE=E
```

Permanent listing files are SAM files containing variable-type records in blocks of 2048 bytes. They remain available until such time as the programmer decides to eliminate them using the ERASE command.

Use of the PARAM Command

The following options of the Executive Command, /PARAM may be used to direct the execution of the COBOL Compiler. The default cases, which are underlined, pertain if the user does not enter a particular parameter.

DEBUG= $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$

If SYMDIC=YES is specified, DEBUG=YES signifies that the source sequence numbers are used to construct COBOL verb symbols. If DEBUG=NO, compiler line numbers are used. If SYMDIC=NO, the DEBUG parameter has no effect.

DISC= $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$

The object module is to be written on the task's object module file (YES), or not (NO).

LIST= $\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$

OBJLST= $\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$

An object listing is to be printed after compilation (YES), or not (NO).

MAP= $\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$

A program map is to be printed after compilation (YES), or not (NO).

XREF= $\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$

A cross-reference list is to be printed after compilation (YES), or not (NO).

DIAG= $\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\end{array}\right\}$

A diagnostic listing is to be printed after compilation (YES) or no diagnostic listing is printed (NO).

CLASS $\left\{\begin{array}{c}\underline{2}\\ 1\end{array}\right\}$

The source program is to be compiled into Class 2 object code (2) or Class 1 object code (1).

SYMDIC= $\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\end{array}\right\}$

An Internal Symbol Dictionary is to be created (YES), or not (NO).

ERRFIL= $\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{YES}\end{array}\right\}$

A diagnostic file is to be created for subsequent examination by BDIAG (YES), or no such file is to be created (NO).

SAVLST= $\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{SOURCE}\\ \text{OBJECT}\\ \text{LOCMAP}\\ \text{ALL}\end{array}\right\}$

A permanent source, object or map listing file is to be created (SOURCE, OBJECT, LOCMAP), or all of these (ALL). NO specifies that a permanent listing file is to be created.

Note: The LIST and SAVLST parameters are independent of one another, that is, LIST=YES need not be specified for SAVLST=SOURCE to be effective.

COBOL LANGUAGE MODIFICATIONS

The COBOL language implemented by the COBOL Background Compilers is a superset of that provided by the respective TOS/TDOS COBOL compilers. The implementation in RCA time sharing systems of various COBOL language elements is described in the paragraphs which follow.

Segmentation

The ANSI COBOL segmentation facility under RCA time sharing systems does not permit the use of fixed, overlayable segments. Fixed, overlayable segments are considered to be permanent segments.

SPECIAL-NAMES Paragraph

The keyword, TERMINAL, may be assigned a mnemonic name.

Although not all the phrases in the SELECT sentence have counterparts in DMS, they are required for syntax capability. The device addressed by a Class 2 object program is determined either from the catalog entry or the FILE command associated with the filename specified in the SELECT sentence. The external name, device class, and unit number designations in the SELECT sentence are for syntactic completeness only. However, other clauses, such as RESERVE, ORGANIZATION, RECORD KEY, etc., are pertinent to DMS FCB operands and must be specified as shown in the COBOL Reference Manual.

Special consideration is given to the system device names when used in the SELECT sentence. FCB's are not created for these files because they are assigned to Executive files as follows: SYSIPT and SYSIN are assigned to SYSIPT; SYSOPT and SYSPUNCH are assigned to SYSOPT; and SYSLST and SYSOUT are assigned to SYSLST.

A read or write to these files generates the appropriate executive macro rather than a DMS input-output macro. The macros issued are the same as those for the ACCEPT and DISPLAY verbs. However, files assigned to the system device names in the SELECT sentence must be programmed as data files, and require OPEN and CLOSE statements as well as AT END imperatives for input files. The advantage of this implementation is that the files are not cataloged, but are EAM files existing only for the duration of the task. They are ideal for low volume input and output.

When the system device names SYSIPT or SYSIN appear in a SELECT sentence, the filename is associated with the Executive's SYSIPT file. The file may be directed to the system card reader or a cataloged SAM or ISAM file by the SYSFILE command.

The record type should be variable and the record size should not exceed 80 bytes for SAM files or 72 information bytes for ISAM files. In ISAM input files the eight-byte key must be the first eight data bytes of the record (after the 4-byte record length field). The key field is placed in positions 73 through 80 of the input record area if more than 72 bytes are defined. The AT END imperative is executed when end-of-file is reached in a cataloged file, or on the first command statement or /EOF card in the card reader.

When the system device names SYSLST or SYSOUT appear in a SELECT sentence, the filename is associated with an Executive EAM file destined to be printed on SYSLST. The size of the record must not exceed the print line size of the system printer or remote batch printer. Because the file is under Executive control and is cumulative throughout the task, the programmer should issue a print control character to skip to the top of page before the first and after the last data records are written to the file by the problem program. The file is printed and then erased at task termination by the control program.

When the system device names SYSOPT or SYSPUNCH appear in a SELECT sentence, the filename is associated with an Executive EAM file destined to be punched as SYSOPT. The size of the record must not exceed 80 bytes. The file is cumulative throughout the life of the task and is punched and erased at task termination by the control program.

The FILE CONTROL Paragraph

SELECT Filename

The name specified is entered in its entirety into the filename operand of the FCB. The name must be a COBOL name; DMS qualified filenames are not permitted. The first eight characters, including hyphens if they appear, of the filename are entered in the FCB as the linkname. Hyphens are valid in the filename of the FCB, but not in the linkname. Therefore, hyphens should be avoided within the first eight characters of the COBOL filename.

External-Name

This entry is required only for syntactical completeness.

UNIT-RECORD

Because Class 2 programs may not access unit-record devices directly, a standard SAM file FCB will be generated for files assigned to unit-record devices. After the problem program has finished executing, the user must issue a PRINT or PUNCH command naming the output file. The FCB of the SAM file specifies fixed, blocked records; such a file could be created by the File Editor or by another problem program.

DIRECT-ACCESS

A standard SAM file FCB is established unless ORGANIZATION IS INDEXED is specified in which case an ISAM file FCB is established.

UTILITY

A standard SAM file FCB is established. However, some changes, such as block size, may be made with a FILE command at run time.

ORGANIZATION

If DIRECT is specified, a SAM file is established in the FCB and user PAM is employed to simulate TOS/TDOS random access processing as defined in the COBOL Reference Manual. Under this mode of operation, a standard PAM page of 2,048 bytes is treated as a track as in TOS/TDOS. The relative track number given in the ACTUAL KEY is interpreted by compiler generated routines as a relative page number. The SYMBOLIC KEY is maintained on the pages as part of the data records. The various access methods defined in the reference manual are simulated by COBOL routines which expect that block size will not exceed 2,048 bytes, less the length of the SYMBOLIC KEY fields. The BLOCK CONTAINS clause is not applicable when ORGANIZATION IS DIRECT, and the BLKSIZE operand in the FCB may not be changed by a FILE command. Files created by COBOL programs under this mode may be processed only by other COBOL programs.

If INDEXED is specified, a DMS ISAM file FCB is established. The compiler will generate routines to process the ISAM files as defined in the COBOL Reference Manual. Any alteration of the OPEN type operand in the FCB by a FILE command is not supported.

If ORGANIZATION is not specified, a standard SAM file FCB is established.

RESERVE

This entry has no effect in class 2 programs because DMS handles buffering.

SYMBOLIC KEY

This entry is treated as defined in the COBOL Reference Manual and is maintained on the file by COBOL generated routines, except when used for ISAM where it is supported by DMS directly, and determines the KEYARG entry in the FCB.

ACTUAL KEY

This entry is processed as defined in the COBOL Reference Manual. However, it is treated as a relative PAM page address rather than a relative track address.

RECORD KEY

This entry determines the FCB operand entries for KEYPOS and KEYLEN. Processing is defined in the COBOL Reference Manual.

The I-O-CONTROL Paragraph

The APPLY RESTRICTED SEARCH OF integer TRACKS ON filename clause is interpreted in Class 2 programs as integer PAGES ON filename though the word TRACKS is required for syntax. In other words, an integer specification of 5 is interpreted as meaning 5 PAM pages (5 standard blocks of 2,048 bytes) rather than 5 physical tracks.

The APPLY BLOCK DENSITY n PERCENT ON filename clause is interpreted as specifying the complement of the percentage to be specified in the PAD operand of the FCB. Thus, 60 PERCENT would cause the PAD factor of 40 to be generated.

The following clauses have no meaning for Class 2 programs in this release: SAME AREA, RERUN, APPLY FORM-OVERFLOW, and APPLY WRITE ONLY, APPLY CHECKPOINT, APPLY CYLINDER-OFLO, and APPLY LOG.

The File Description (FD) Paragraph

Recording

The recording mode specified will be used in the FCB as long as it is syntactically acceptable to the compiler. The use of undefined records, however, is highly inefficient because DMS will write them as one record per standard 2048-byte block. The BLOCK CONTAINS clause cannot override this restriction.

Fixed, blocked records will be specified in the FCB if F is given. However, because system products and Executive system files all use variable-type records, an FCB which specifies fixed-type records cannot handle files created by the DATA command etc. Nor can a system product read a file created with such an FCB.

The most efficient recording mode is V and is to be preferred over fixed and undefined.

The compiler always generates the BLKSIZE operand of the FCB in terms of a number of standard blocks of 2,048 bytes. For standard SAM files, only 2,044 bytes are available for data. For ORGANIZATION IS DIRECT, the full 2,048 bytes are available.

If the BLOCK CONTAINS clause specifies more than 2,044 bytes, the number of standard blocks required to contain that number of bytes is requested in the FCB. Therefore, if BLOCK CONTAINS 2045 CHARACTERS is specified, a buffer of 4,096 bytes will be requested and 2,043 bytes of each block written will be unused. It is the programmer's responsibility to determine what multiple of 2,044 bytes is required to make the most efficient use of the space available. The default, if BLOCK CONTAINS is not specified, is one standard block. For magnetic tape output files, the BLKSIZE operand of the FCB may be changed at run time by a FILE command. If other than standard blocks are written or read to tape, the BLKSIZE must be specified in the FILE command.

Record Contains

This entry is interpreted as specified in the COBOL Reference manual.

Label Records

STANDARD must be specified for all but UNIT-RECORD and magnetic tape files. For UNIT-RECORD files, the FCB entry will be changed to STANDARD by the compiler.

Magnetic tape files with UTILITY specified in the SELECT sentence may have STANDARD, OMITTED, or data name specified and they will be processed as, defined in the COBOL Reference Manual.

Value of ID

This is not supported. DMS does not allow the user program to access the HDR1 label. Only UHL and UTL labels may be created and checked by user programs.

The ACCEPT Statement

COBOL Statement	Macro Issued
ACCEPT data-name	RDCRD
ACCEPT data-name FROM CONSOLE	TYPIO
ACCEPT data-name FROM TERMINAL	RDATA
ACCEPT data-name FROM SYSIN	RDCRD
ACCEPT data-name FROM SYSIPT	RDCRD

Note: When TERMINAL is used in a nonconversational program a RDCRD is issued.

The DISPLAY Statement

COBOL Statement	Macro Issued
DISPLAY data-name	PROUT
DISPLAY data-name UPON CONSOLE	TYPIO
DISPLAY data-name UPON TERMINAL	
DISPLAY data-name UPON SYSPUNCH	WRTOT
DISPLAY data-name UPON SYSOPT	WRTOT
DISPLAY data-name UPON SYSOUT	PROUT
DISPLAY data-name UPON SYSLST	PROUT

Note: When TERMINAL is used in a nonconversational program a PROUT is issued.

Debugging Language Statements

The TRACE and EXHIBIT verbs write to the system printer.

STOP Verb

The STOP RUN statement generates a QUIET macro followed by a TERM macro. The STOP 'literal' statement will issue a TYPIO macro to print the literal. The macro requires a reply in this instance. The reply may be a null message (EOT key only), or any other message from the console. The message is not available to the program, but does cause execution to be resumed.

COBOL INTERFACE WITH DMS AND OTHER SYSTEM COMPONENTS IN HANDLING FILES ACCESSED BY OBJECT PROGRAMS

Access Methods

COBOL object programs use the following Data Management System access methods:

SAM	(Sequential Access Method)
ISAM	(Indexed Sequential Access Method)
PAM	(Direct Access Method)

The selection of an access method is based on the ORGANIZATION and ACCESS clauses in the SELECT sentence of the Environment Division.

File Description

The user may wish to make his input and output files acceptable to other system components. For instance, he might wish to use the /DATA command which creates a file containing standard blocks of variable type. The programmer may wish to take this into consideration when he is planning his file description (FD).

FILE Command

Before executing an object program the programmer must issue a FILE command with the LINK=linkname parameter (refer to Data Management System Reference Manual) for each file which is to be accessed by that object program. The link name connects the filename in the user's catalog with the filename in the COBOL File Description. To ensure correct use of the link feature, the programmer must be careful that:

1. The first eight characters of the FDname are unique among the first eight characters of all other filenames in his program.

2. The character – (hyphen or dash) does not appear in the first eight characters of the FDname.

3. The FILE command must not be used to modify any of the following items in the FCB in the COBOL object program:

FCBTYPE	KEYLEN	INDEX
RECFORM	KEYPOS	LABEL
VARBLD	PAD	OPEN
OVERLAP	DUPKEY	

If the programmer were to use the FILE command to override any of the above for a COBOL object time file, the results would be unpredictable.

In addition, if the object program is to write to an output file which has not yet had space allocated to it, the FILE command must also include the SPACE=(primary-integer, secondary-integer) parameter.

Device Independence for Sequential Files

Although the programmer specifies DIRECT-ACCESS or UTILITY in the ASSIGN clause, this clause can be overriden by the appropriate parameter in the FILE command at execution time.

COBOL SOURCE LIBRARY ON DISC

The COBOL Compiler supports the use of a disc resident COBOL source program library. The programmer creates and maintains a source library by using the COBOL Library Update Routine (COBLUR). For a full description of this routine, refer to the Utility Routines Reference Manual.

If the source program which is supplied as an input to the compiler contains BASIS, INCLUDE or COPY statements, the compiler will take the specified program or portion of the program from the specified source library file.

COBLUR creates an ISAM file with variable length records indexed by a 14-byte alphanumeric key which is composed of the library-entry name (8 bytes) and the six digit source sequence number of the source coding. The ISAM library file consists of four sections:

Section 1	Identification/Environment Entries
Section 2	Data Division Entries
Section 3	Procedure Division Entries
Section 4	Complete Source Program Entries

Because the source library file has records with alphanumeric keys, it cannot be accessed by the File Editor, which handles only records with numeric keys.

When the programmer wishes the compiler to retrieve all, or part, of a source program from the source library, he must issue a FILE command for the library file before invoking the compiler. In this FILE command, the link name must be specified as COBLIB. Thus, if the programmer has a COBOL Source Library file named SOURCLIB, he must enter the command:

```
/FILE SOURCLIB,LINK=COBLIB
```

Note: The library-entry name specified in the BASIS, INCLUDE and COPY instructions is an external name, therefore, in RCA COBOL, it must be enclosed in apostrophes.

DIAGNOSTIC FILE GENERATION

The COBOL Compiler creates a permanent disc resident diagnostic file if the programmer so requests via the command:

```
/PARAM ERRFIL=YES
```

before compilation.

The file contains English language error messages describing each COBOL source error detected during compilation. The programmer at a remote terminal can subsequently invoke the Post-Compilation Diagnostic Routine to access his error file and display the messages therein. For a full description of the Post-Compilation Diagnostic Routine, see Chapter 4 of this manual.

The COBOL Compiler creates a catalog entry and allocates space for the diagnostic file if the programmer has not already done so. It constructs a filename by prefixing the source program name (not more than the first eight characters) to the basename ERRFIL, thus:

source-program-name.ERRFIL

GENERATION OF INTERNAL SYMBOL DICTIONARY (ISD)

The COBOL Compiler generates an Internal Symbol Dictionary (ISD) for the COBOL program if the user so requests at compilation time via the command:

```
/PARAM SYMDIC=YES
```

The Internal Symbol Dictionary records are contained in the output object module.

The ISD extends the programmer's debugging capabilities when he is checking the logic of the COBOL object program with IDA. With an ISD, the programmer will be able to specify COBOL source data names, procedure names and verbs in IDA commands at object program execution time; without it, the programmer would have to refer to them as hexadecimal addresses within the object program.

If the SYMDIC parameter is specified, the compiler examines the Data Definitions and Procedure Descriptors to collect information describing data name and procedure name items. It creates an ISD definition for each such item.

The COBOL compiler creates an entry in the ISD records for every data name in the DATA DIVISION and procedure and section names in the PROCEDURE DIVISION. The data names from the FILE SECTION of the DATA DIVISION must not be referenced with IDA statements until after the file has been opened and, in the case of input files, a record has been read. The compiler lists the names of all symbols for which there is an entry on the ISD. This list is included in the object listing. A COBOL symbol is written in an IDA statement within delimiting apostrophes, e.g., 'START-PARA'. COBOL qualification may be included as part of the symbol along with the key words OF and IN. The COBOL qualifier is written within the apostrophes, for example: 'DAY OF MONTH IN MASTER-RECORD'. IDA qualifiers, separating periods, offsets, subscripts, etc. are written outside the apostrophes. Any item described by an OCCURS clause may be subscripted in IDA by numeric constants for up to three dimensions. Symbols or expressions may not be used for subscripts. The subscripts must immediately follow the closing apostrophe, be enclosed in parentheses, and be separated by commas, for example: 'TABLE-ITEM' (3,9).

COBOL also generates IDA symbols for each verb in the Procedure Division, by statement number. The format for referencing verbs through IDA is as follows:

'statement-number verb position'

statement-number

Is the source sequence number or compiler line number of the source statement generated by the compiler, depending on whether the value of the DEBUG= was YES or NO, respectively.

verb

Is the verb on that line which is to be referenced. (The verb name may require abbreviation For a list of abbreviations, please refer to the IDA Reference Manual).

position

Is the sequential position of the verb on the line in the instance where the same verb is used more than once on the same source line. A blank indicates the first occurrence, the digit 2 the second occurrence, the digit 3 the third occurrence, etc.

Example:

Source line 600210	MOVE A TO B, MOVE A TO C.
IDA Reference '600210MOV '600210MOV2'	 (first occurrence) (second occurrence)

SAMPLE SESSIONS

Example A

The following is a sample session in which a programmer at a remote terminal compiles a COBOL source program, stores the object module in an Object Module Library, loads the object program, enters IDA commands, and executes the object program.

```
    %C E222 PLEASE LOGON.
 1  /LOGON KREFTEST
    %C E223 LOGON ACCEPTED AT 1009 ON 04/22/70, TSN 1245 ASSIGNED.
 2  /PARAM LIST=YES,CLASS=2,SYMDIC=YES
 3  /SYSFILE SYSDTA=EKCOBSRI
    32A0 COMPILATION INITIATED (BGCOB VERSION=035B)
    32AA COMPILATION COMPLETED
 4  /EXEC BGCOB
    %L001 PROGRAM LOADING
    %EB001 SPOOLOUT INITIATED FOR TSN=1246 ID=KREFTEST
    %      PRINT FILE=00004
 5  /FILE OMLIB,SPACE=(18,3)
    /FSTATUS OMLIB
    %0000018 OMLIB
 6  /SYSFILE SYSDTA=(SYSCMD)
 7  /EXEC (LMR)
    P001 - DLL V-2A
    *CONTROL OUTFILE=OMLIB
    *ADD SOURCE=*,OBJMOD=TSTISAM
    *END
    LM 039 NORMAL JOB TERM.
 8  /PUNCH *
    %EB001 SPOOLOUT INITIATED FOR TSN=1248 ID=KREFTEST
    %      PUNCH FILE=00005
    /FILE EKISAM,LINK=EKISAM
    /FILE OUTFILE2,LINK=OUTFILE2,SPACE=(6,3)
 9  /LOAD (TSTISAM,OMLIB),IDA=YES
    %P001 - DLL V-2A
10  /AT 'PARA2';/D 'WA1'
    /RESUME
11  %INTERRUPTED AT 001A9A
    % VM-ADR 001698:00169C P-COUNT 001A9A
    % 001698    00000.
    /R
    %INTERRUPTED AT 001A9A
    % VM-ADR 001698:00169C P-COUNT 001A9A
    % 001698    00001.
12  /STOP
    /LOGOFF
    %C E420 LOGOFF AT 1017 ON 04/22/70, FOR TSN 1245
    %C E421 CPU TIME USED: 0018.4160 SECONDS.X
```

Notes on Example A:

(1)	Programmer logs on.
(2)	Indicates compilation-time parameters. In this case, a source-program listing and a Class 2 object module containing Internal Symbol Dictionary records.
(3)	Directs SYSDTA to the cataloged disc file which contains the source language program.
(4)	Invokes execution of the COBOL compiler.
(5)	Create a catalog entry and allocate space for an Object Module Library file.
(6)	Re-direct SYSDTA to the terminal so that the programmer may type in instructions to the Library Maintenance Routine.
(7)	Invoke Library Maintenance Routine and enter the instructions required to create an Object Module Library.
(8)	Punch an object deck. Note: This erases the Object Module File.
(9)	Load the object module from the library. IDA=YES must be specified if the user intends to enter IDA commands.
(10)	Enter an IDA command, specifying COBOL source names.
(11)	IDA interrupts execution of object program and displays the value of the specified data-name.
(12)	Programmer stops execution of object program and logs off.

Example B

Conversational user executing a compilation and linking his object module.

```
   /LOGON KREFTEST
   %C E223 LOGON ACCEPTED AT 1042 ON 05/05/70, TSN 0310 ASSIGNED
①  /PARAM LIST=YES,OBJLST=YES,XREF=YES,CLASS=2
②  /SYSFILE SYSDTA=EKCOBSRI
③  /EXEC BGCOB
   %L001 PROGRAM LOADING
   32A0 COMPILATION INITIATED (BGCOB VERSION = 035B)
④  32AA COMPILATION COMPLETED
   %EB001 SPOOLOUT INITIATED FOR TSN=0314 ID=KREFTEST
   %      PRINT FILE=00006
⑤  /SYSFILE SYSDTA=(SYSCMD)
   /EXEC TSOSLNK
   *PROGRAM LNKCOB,XREF=Y
    LIST?  N
    PROGRAM LNKCOB,XREF=Y
   *INCLUDE *
    INCLUDE *
   *END
    END
⑥   PROGRAM BOUND
⑦   PROGRAM WRITTEN
   /LOGOFF
```

Notes on Example B:

①	Programmer specifies parameters for COBOL compiler.
②	and directs SYSDTA to COBOL source file.
③	Executes compilation.
④	32AA code indicates an error-free compilation.
⑤	Programmer re-directs SYSDTA to the terminal in order to type Linkage Editor statements.
⑥&⑦	Linkage Editor messages indicating successful executing.

Example C

Conversational user compiling a COBOL program and executing it immediately.

```
① /PARAM LIST=YES,CLASS=2
② /SYSFILE SYSDTA=EKCOBSRI
③ /EXEC BGCOB
  %L001 PROGRAM LOADING
   32A0 COMPILATION INITIATED (BGCOB VERSION=035B)
   32AA COMPILATION COMPLETED
  %EB001 SPOOLOUT INITIATED FOR TSN=0147 ID=KREFTEST
  %      PRINT FILE=00005
④ /FILE EKISAM,LINK=EKISAM
  /FILE OUTFILE2,LINK=OUTFILE2,SPACE=(6,3)
⑤ /EXEC *
   P -DLL V-2A
  %P001 DYNAMIC LOADER INVOKED
⑥ /PRINT EKISAM
  %EB001 SPOOLOUT INITIATED FOR TSN=0148 ID=KREFTEST
  %      PRINT FILE=EKISAM
  /PRINT OUTFILE2
  %EB001 SPOOLOUT INITIATED FOR TSN=0149 ID=KREFTEST
  %      PRINT FILE=OUTFILE2
```

Notes to Example C:

①	Programmer describes parameters required – a source listing to be printed and a Class II object module. (Note that only Class II object modules are acceptable to the Dynamic Linking Loader.)
②	Direct SYSDTA to source file.
③	Invoke the COBOL compiler.
④	FILE command must be issued to define files accessed by object program.
⑤	Execute object program directly from EAM object module file. Note that the Dynamic Linking Loader creates only temporary load modules.
⑥	The programmer may wish to spool-out input and output files to check on the execution of the object program.

2. FORTRAN BACKGROUND COMPILATION

INTRODUCTION

The FORTRAN Background Compiler is a nonconversational Class II (pageable) program which runs under control of RCA time-sharing operating systems.

The FORTRAN Compiler is a disc-oriented program; it accepts source input from disc or card files and generates an object module on disc.

EQUIPMENT CONFIGURATION

The FORTRAN Background Compiler requires approximately 100,000 bytes of virtual memory.

The following minimum configuration is required for a FORTRAN compilation:

1 – Processor (262KB)

2 – Disc Storage Units

1 – Paging Drum

In addition, the system must include a Communications Controller Multichannel (CCM) with the associated buffers and remote terminals if it is to support conversational users.

Optional Equipment:

1-card-reader: if input is to come from cards.

1-card-punch: if the object module is to be punched as cards.

1-printer: if listings are required.

SOURCE LANGUAGE COMPATIBILITY

The language elements implemented by the FORTRAN Background Compiler are the same as those provided in the TOS/TDOS Compiler. In addition, FORTRAN programs prepared by the Interactive FORTRAN System (IFOR) are acceptable for compilation by the compiler.

SOURCE INPUT

The FORTRAN Background Compiler reads source input by using the RDATA macro. RDATA causes the Executive to read an input record from the SYSDTA file. The SYSDTA file may be a cataloged file, a card reader, or a remote terminal. (See figure 2-1.)

Disc-Resident Source Files

A cataloged FORTRAN source file must be either a Sequential Access Method (SAM) file or an Indexed Sequential Access Method (ISAM) file. In an ISAM file, the 8-byte numeric keys must be the first 8 data bytes of the record. ISAM files may be directly created, updated, or edited by the File Editor (see the File Editor Reference Manual). File Editor can also edit SAM files indirectly. The user may also use the Interactive FORTRAN System (IFOR) to prepare a FORTRAN source program, which will be free of syntactical errors, as input to the compiler.

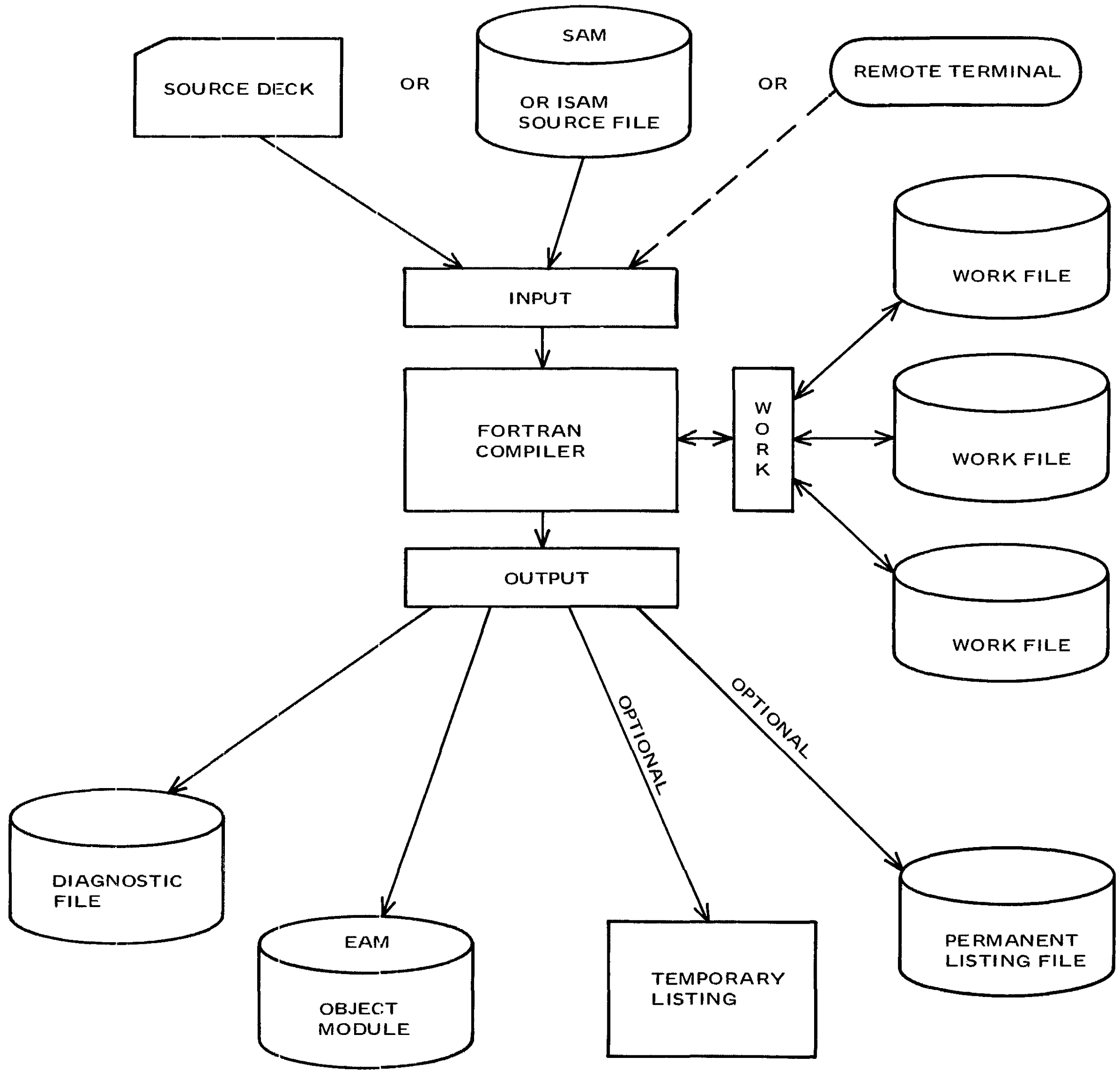


FIGURE 2-1. FORTRAN BACKGROUND COMPILER INPUT-OUTPUT FLOWCHART

Before he initiates a compilation, the user must direct SYSDTA to the source file with the command:

```
/SYSFILE SYSDTA=filename
```

where filename is the cataloged name of his file.

> Note: If the user wishes to do additional processing after the end of the compilation, for example, Library Maintenance Routine, he must redirect SYSDTA from his FORTRAN source file to whatever source is required by subsequent routines. For instance, a conversational user wishing to redirect SYSDTA to his terminal would issue the following command:
>
> ```
> /SYSFILE SYSDTA=(SYSCMD)
> ```

Card Input

A user may enter his entire compilation as a background task in the form of a card deck containing a /LOGON card, cards containing commands to the compiler, the FORTRAN source program, and a /LOGOFF card. The operator places the deck in the reader and initiates execution of the task by entering the RCARD command on the console, and the entire card deck is spooled in and then executed. In this case, the user should direct SYSDTA so that the compiler can take its input from the spooled-in file, thus:

```
/SYSFILE SYSDTA=(SYSCMD)
```

In a conversational task, SYSDTA is, by definition, the user's terminal. A conversational user initiating a compilation from his terminal may instruct the compiler to take its input from a source deck which he had previously requested the operator to place in the reader. In this case, he should redirect SYSDTA from the terminal to the card reader by giving the command:

```
/SYSFILE SYSDTA=(CARD)
```

The operator then receives a message asking him to confirm that the deck is actually in the reader. SYSDTA should be redirected from the card reader as soon as possible to free it for system use.

INTERMEDIATE WORK FILES

The FORTRAN compiler uses direct-access files for its intermediate work space. The compiler accesses these disc files by using the DMS EAM function.

GENERATED OBJECT MODULES

The compiler generates object modules on a direct-access EAM file from which the user may punch it onto cards using the spoolout PUNCH command.

Disc-Resident Object Modules

The compiler automatically writes the object module on an EAM file unless the user specifically instructs it not to do so by means of the following command:

```
/PARAM DISC=NO
```

Because files created by EAM are temporary (at most, they remain available for the duration of the task in which they are created), the user who wishes to preserve his object module should do so by invoking the Library Maintenance Routine (for a description of this routine, refer to the Utility Routines Reference Manual).

Object Module Card Decks

If the user wishes to receive an object module in the form of a deck of punched cards, he must issue a

```
/PUNCH*
```

command after the end of compilation and before logging off. This instructs the system to punch out the EAM object module file; thus, the user who wants an object deck must not have previously issued a command:

```
/PARAM DISC=NO
```

which would inhibit the creation of an EAM object module file.

Furthermore, once the PUNCH command has been issued, the EAM object module file will no longer be available to the user. If he wishes to save the object module in a library, he should do so by executing the Library Maintenance Routine before issuing the PUNCH command.

Class I and Class II Object Modules

The compiler can produce either Class I or Class II object modules. The user should specify the type of object module he requires in the /PARAM command, as follows:

/PARAM CLASS=1
: produces Class I (nonpageable) programs

/PARAM CLASS=2
: produces Class II (pageable) programs

If the user does not specify the CLASS parameter, the default option of 2 pertains.

Class I object module output contains external references to TOS/TDOS FORTRAN Call Library modules; the program should be bound and executed as a Class I program. Class II object module output contains external references to the TSOS FORTRAN Object Module Library; the program should be bound and executed as a Class II program.

GENERATED LISTINGS

Immediate Listings

The FORTRAN user may instruct the compiler to produce a source program listing by specifying the parameter LIST=YES in the /PARAM command.

A diagnostic listing is always printed if any diagnostics exist.

An object program map will be printed unless the user suppresses its printing by specifying the parameter MAP=NO in the /PARAM command.

The compiler generates these three listings on a temporary, EAM, direct-access file and at the end of compilation places them on the spoolout queue to be printed. After they have been spooled out, they are destroyed.

Saved Listings

The user may have his listings written on cataloged SAM files; in this case, they are not automatically printed. However, the files are available for the user to access at any time until he himself chooses to ERASE them.

The compiler will write any or all of the following listings on cataloged files if the user specifies the following parameters in the /PARAM command:

SAVLST=SOURCE
: specifies that the source program listing is to be written on a cataloged file.

SAVLST=LOCMAP
: specifies that the object-program map is to be written on cataloged file.

SAVLST=ALL
: specifies that both of the above are to be written on cataloged files.

The compiler constructs filenames for these files by prefixing the FORTRAN source program name to the base-name SAVLST, e.g.,

source-program-name.SAVLST

The filename is further qualified by suffixing the base-name SAVLST with a selected mnemonic resulting in one of the FORTRAN-specified combinations:

source-program-name.SAVLST.S
: for source listing file

source-program-name.SAVLST.M
: for map file

INTERNAL SYMBOL DICTIONARY (ISD) GENERATION

The FORTRAN compiler generates an Internal Symbol Dictionary (ISD) for the FORTRAN program if the user so requests at compilation time in the command:

```
/PARAM SYMDIC=YES
```

The ISD extends the user's debugging capabilities when he is checking out the FORTRAN object program at execution time using IDA. With an ISD, the user will be able to specify his FORTRAN source variable names and statement numbers in IDA commands at object program execution time; without it, he would have to refer to them as hexadecimal addresses within his object program.

If the SYMDIC parameter is specified, the compiler generates ISD definitions defining each symbol's object program location, length, and type attributes.

The Linkage Editor and Linking Loader can generate ISD symbol entries for every entry symbol appearing in the ESD records for use by the IDA system. The generation of this data is distinct from the compiler's generation of ISD records and must be specifically requested. The Linkage Editor and Linking Loader generate ISD definitions for the program name, named COMMON, and label definitions (ESD types SD, CM, and LD, respectively).

ISD information collected by the compiler is passed along in the form of ISD records in the output object module.

DIAGNOSTIC FILE GENERATION

The FORTRAN compiler creates a permanent disc-resident diagnostic file if the user so requests via the command:

```
/PARAM ERRFIL=YES
```

before compilation.

The file contains English-language error messages describing each FORTRAN source error detected during compilation. The conventional user at a remote terminal can subsequently invoke the Post-Compilation Disgnostic routine to access his error file and display the messages therein. For a full description of the Post-Compilation Diagnostic routine, see Chapter 4 of this manual.

The FORTRAN compiler creates a catalog entry and allocates space for the diagnostic file if the user has not already done so. It constructs a filename by prefixing the source-program name to the base-name ERRFIL thus:

```
source-program-name.ERRFIL
```

The user may use the FILE command of DMS to create a catalog entry and to allocate space for his diagnostic file before executing his compilation. In this case, the filename may be the name which the compiler would construct or a user-assigned name. In either case, he must specify the linkname of ERRFIL in his FILE command, for example,

```
/FILE filename, LINK=ERRFIL,SPACE=(primary-integer, secondary
integer)
```

SUMMARY OF FORTRAN COMPILER PARAMETER SPECIFICATIONS

The options listed below for the /PARAM command may be used to direct the execution of the FORTRAN compiler. The default cases, which are underlined, pertain if the user does not enter a particular parameter.

CARD=		
	YES	The object module is written to the task's EAM object module file on disc for subsequent punching when the programmer enters the PUNCH* command. Note: This option need not be used if DISC=YES is specified, since the same file will be written in either case.
	NO	The object module is not written to the EAM object module file unless DISC=YES is in effect.
CLASS=		
	1	The source program is to be compiled into Class I (nonpageable) object code for TOS-compatible execution.
	2	The source program is to be compiled into Class II (pageable) object code for execution under the TSOS control program.
CODE=		
	1	The source program records are to be read as EBCDIC characters.
	2	The source program records are to be read as 7094-compatible characters.
	3	The source program records are to be read as 3301-compatible characters.
DEBUG=		
	YES	The execution-time diagnostic routine is to produce source program line number references in the error analysis listing.
	NO	The line number reference facility is not to be used.
DISC=		
	YES	Object modules are to be written to the task's EAM object module file on disc for subsequent processing as filename *. This option is redundant when CARD=YES has been specified.
	NO	Object modules are not to be written to the task's EAM object module file unless CARD=YES has been specified.

ERRFIL=		
	YES	All diagnostic messages along with a diagnostic summary record are to be written to a SAM file named progname.ERRFIL for subsequent interrogation using the BDIAG routine. Progname is the name of the user's program, subprogram or function subprogram and if not specified will be NONAME by default; that is, NONAME.ERRFIL. The user's file will be used if a FILE command specifying ERRFIL as the linkname is issued before the EXECUTE or RESUME command.
	NO	The diagnostic SAM file is not to be written.
LIST=		
	YES	A source program listing is to be written to the EAM output file and printed when the compilation process is completed. The file is erased after printing.
	NO	A source program listing is not to be written to the EAM output file.
MAP=		
	YES	The program's location map listings are to be written to the EAM output file and printed when the compilation process is completed. The file is erased after printing.
	NO	The locator map listings are not to be written to the EAM output file.
SAVLST=		
	SOURCE	A listing of the source program is to be written to a SAM file for post-compilation printing. The command used for printing the file is /PRINT progname.SAVLST.S,SPACE=E. The filename is progname.SAVLST.S and the linkname is SAVLST.
	LOCMAP	Listings of the locator map are to be written to a SAM file for post-compilation printing. The command used for printing the file is /PRINT progname.SAVLST.M,SPACE=E. The filename is progname.SAVLST.M and the linkname is SAVLST.
	ALL	Both the source program and locator map listings are to be written to a SAM file for post-compilation printing. The command used to print the file is /PRINT progname.SAVLST.A,SPACE=E. The filename is progname.SAVLST.A and the linkname is SAVLST.
	NO	The SAVLST file is not to be created.

SYMDIC=

YES	An Internal Symbol Dictionary is to be created with an entry for each variable name, statement number, and line number of executable instructions. The ISD records are to be incorporated into the object module for use by symbolic IDA.
<u>NO</u>	ISD records are not to be generated.

FORTRAN OBJECT TIME FACILITIES

FORTRAN and DMS Related Concepts

A file is said to be stored in the system if it resides on one or more direct-access or magnetic tape devices (volumes) and if the identification of these volumes (volume serial number) is available in the system catalog.

A volume may be classified as either public or private. A public volume is a direct-access volume and must be mounted and on-line during the entire period of system operation. A public volume may be used by many tasks concurrently. A private volume need not be mounted for the entire period of system operation. Its use is restricted to one task at a time and it need only be mounted when the task refers to it. A direct-access volume may be a private volume, while magnetic tape volumes are always classified as private volumes.

Files to be cataloged can be stored on all types of volumes. Files used within a task need not be cataloged if their use is temporary (that is, confined to the task); the FORTRAN EAM data set could be utilized here. Users generally make the most effective use of the system by storing their files on public volumes and cataloging them, when it is necessary to retain the files in the system. Public volumes are always on line, and files stored on public volumes are always available for access to a user's task.

A user can allocate space for his files on public volumes within the limits of public space allocation established for him at JOIN time.

If a user employs private volumes, he may need to wait for the devices on which to mount the volumes, because each time a request is made for a device on which to mount a private volume, the system must determine whether or not it can honor the request.

When each user is joined to the system, the system controller specifies the maximum amount of space on public volumes which the user may acquire. The space, which is referred to as the user's public space allotment, is not actually allocated to the user until he requests it.

For files which reside on public volumes, the system will dynamically allocate space whenever this is required, and if necessary, will acquire space from other public volumes. The important attributes of public volumes are summarized as follows:

1. The public volume is the default type of volume.

2. A given file may be contained on any number of public volumes without the user's knowledge; in particular, it is the only type of volume on which a file can be extended across volumes during the execution of the problem program.

3. Full file security and integrity are provided, yet the volume may be used concurrently by any number of tasks.

4. The volume is always mounted (accessible).

Compilation and Execution Considerations

The FORTRAN user can achieve volume (device) independence, eliminate the need to reestablish the standard data definition, and organize the execution procedure for a variety of tasks.

Volume or device independence can be achieved by assigning integer variables as the data set reference numbers in the FORTRAN source program. This is particularly useful in allowing the entering of input data from a number of sources (volumes with different formats).

Facilities provided by the operating system include the following:

1. Class II programs which do not contain an overlay structure can be executed immediately after compilation. The user should issue the Executive command /EXEC *, thus invoking the Dynamic Linking Loader. Class I programs and those which contain an overlay structure must be bound by the Linkage Editor before being executed. (For a full description of the Linkage Editor and the Static and Dynamic Linking Loaders, refer to the TSOS Utility Routines Reference manual.)

2. Class II object programs are pageable; they require less main memory and therefore more programs can run concurrently.

3. Files may be put on public volumes. Such files do not require mounting and dismounting of devices; the sharing of volumes permits more programs to be run concurrently.

FORTRAN Data Management Facilities

File management facilities provide the means for identifying files; for storing and retrieving them within the system; for sharing them with other users; for copying, modifying and erasing them; and for defining their existence and use in the system. Problem program input/output facilities provide for the actual transfer of data to and from programs which are in execution.

The creation and specification of the data sets is derived from two sources:

1. The FORTRAN Compiler I/O Language Elements.
2. The user designated data set definition.

FORTRAN Compiler Data Set Specification

The FORTRAN I/O statements identify the basic input/output functions at the logical level through format statements and READ/WRITE lists. The physical implementation details, such as the allocation of memory (buffer) and device resources, are either system supplied or supplied through a user-defined data set reference. The link between the physical implementation and the FORTRAN I/O statement is the data set reference number. The data set reference number is contained in a Device List Table linking the reference number to the device characteristics.

SEQUENTIAL DATA SET SPECIFICATION

In FORTRAN, records for sequential data sets are defined by specifications in FORMAT statements and by READ/WRITE lists. A record defined by a specification in a FORMAT statement is a FORTRAN record. A record defined by a READ/WRITE list is a logical record. Within each category, there are three types of records: fixed-length, variable-length, and undefined. In addition, fixed-length and variable-length records can be blocked.

RECORDS CONTROLLED BY FORMAT STATEMENT

Fixed-Length Undefined Records:

For fixed-length and undefined records, the record length and buffer length are specified in the BLKSIZE parameter. For variable-length records, the record length may be specified in the RECSIZE parameter; the buffer length, in the BLKSIZE parameter. The information coded in a FORMAT statement indicates the FORTRAN record length (in bytes).

Fixed-Length Unblocked Records:

For unblocked fixed-length records written under FORMAT control, the FORTRAN record length must not exceed BLKSIZE (see figure 2-2).

Example: Assume BLKSIZE=44

```
10          FORMAT(F10.5,I6,2F12.5,'SUMS')
            WRITE(20,10)AB,NA,AC,AD
```

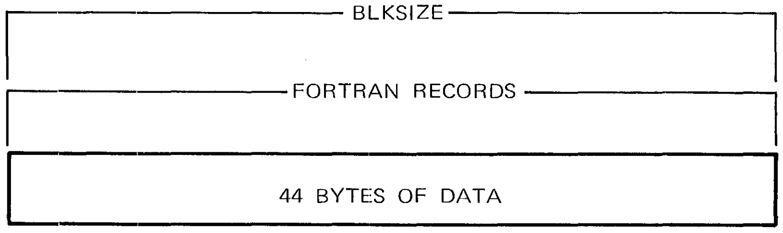


FIGURE 2-2. FORTRAN RECORD (FORMAT CONTROL) FIXED-LENGTH SPECIFICATION

If the FORTRAN record length is less than BLKSIZE, the record is padded with blanks to fill the remainder of the buffer (see figure 2-3). The entire buffer is written.

Example: Assume BLKSIZE=56

```
5          FORMAT(F10.5,16,F12.5,'TOTAL')
           WRITE(15,5)BC,NB,BD
```

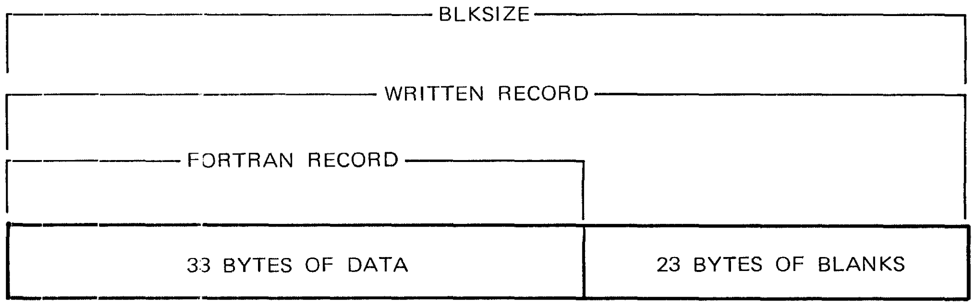


FIGURE 2-3. FORTRAN RECORD (FORMAT CONTROL) WITH FIXED-LENGTH SPECIFICATION

Variable-Length Unblocked Records:

For unblocked variable-length records written under FORMAT control. BLKSIZE is specified as eight greater than the maximum FORTRAN record length. These extra 8 bytes are required for the 4-byte block control word (BCW) and the 4-byte segment control word (SCW), as shown in figure 2-4. The BCW contains the length of the block; the SCW contains the length of the record segment (denoted by LRECL); i.e., the data length plus 4-bytes for the SCW.

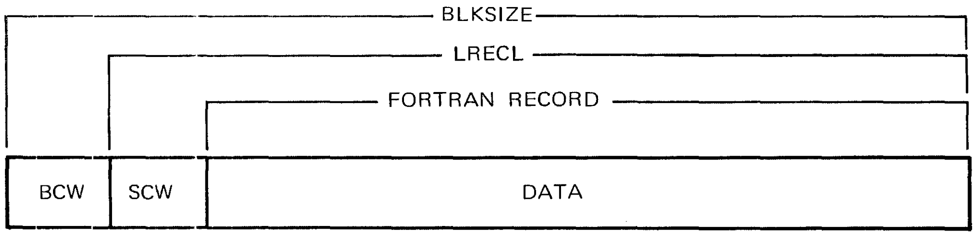


FIGURE 2-4. FORTRAN RECORD (FORMAT CONTROL) WITH VARIABLE-LENGTH SPECIFICATION

If the FORTRAN record length is less than BLKSIZE, the unused portion of the buffer is not written (see figure 2-5).

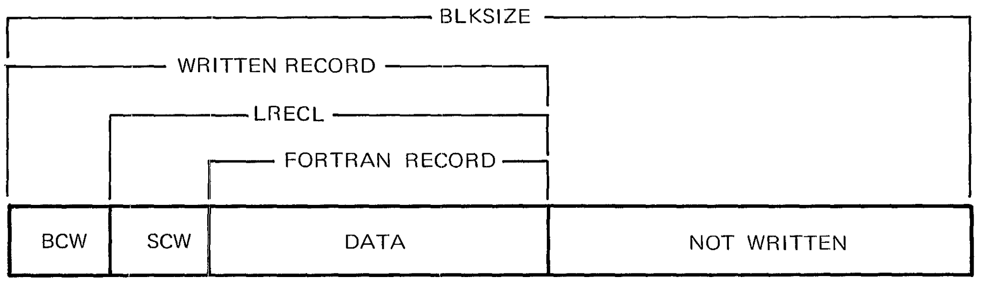


FIGURE 2-5. FORTRAN RECORD (FORMAT CONTROL) WITH VARIABLE-LENGTH SPECIFICATION

Undefined Records:

For undefined records written under FORMAT control, BLKSIZE is specified as the maximum FORTRAN record length. If the FORTRAN record length is less than BLKSIZE, the unused portion of the buffer is not written (see figure 2-6).

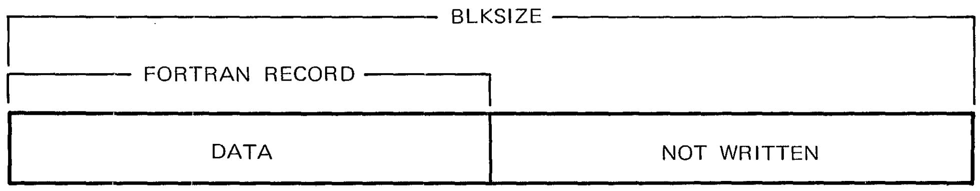


FIGURE 2-6. FORTRAN RECORD (FORMAT CONTROL) WITH UNDEFINED SPECIFICATION

Blocked Records:

For all blocked records, the record length is specified in the RECSIZE parameter; the block length and buffer length in the BLKSIZE parameter.

Blocked fixed-length records:

For blocked fixed-length records written under FORMAT control, RECSIZE is specified as maximum possible FORTRAN record length, and BLKSIZE must be an integral multiple of it. If the FORTRAN record length is less than LRECL, the rightmost portion of the record is padded with blanks (see figure 2-7).

Example: Assume BLKSIZE=48 and LRECL=24

```
10          FORMAT(I8,F16.4)
20          FORMAT(I12)
               .
               .
               .
            WRITE(13,10)N,B
               .
               .
               .
            WRITE(13,20)K
```

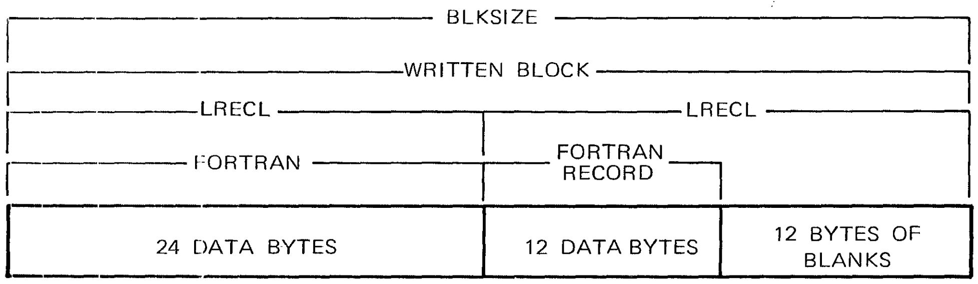


FIGURE 2-7. FIXED-LENGTH BLOCKED RECORDS WRITTEN UNDER FORMAT CONTROL

Blocked Variable-Length Records:

For K-blocked variable-length records written under FORMAT control BLKSIZE must be equal to or greater than K (LRECL); where the value of K is the number of records in a block.

Four additional bytes allocated with BLKSIZE are required for the block control word that contains the block length. The 4 additional bytes allocated with LRECL are used for the segment control word that contains the record-length indicator.

If a WRITE is executed and the amount of space remaining in the present buffer is less than LRECL, only the filled portion of this buffer is written; the new data goes into the next buffer. However, if the space remaining in a buffer is greater than LRECL, the buffer is not written, but held for the next WRITE (see figure 2-8). If another WRITE is not executed before the job step is terminated, then the filled portion of the buffer is written.

Example: Assume BLKSIZE=28 and RECSIZE=8

```
30          FORMAT(I3,F5.2)
40          FORMAT(F4.1)
50          FORMAT(F7.3)
               .
               .
               .
```

```
WRITE(12,30)M,Z
.
.
.
WRITE(12,40)V
.
.
.
WRITE(12,50)Y
```

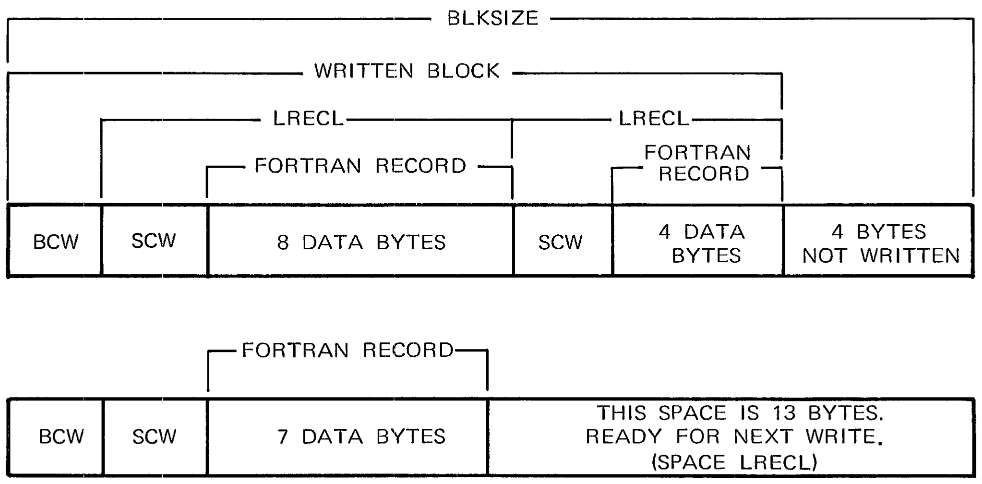


FIGURE 2-8. VARIABLE-LENGTH BLOCKED RECORDS WRITTEN UNDER FORMAT CONTROL

Logical Records (Not Controlled by FORMAT Statement)

All record formats specified for FORMAT-controlled sequential data sets are supported for sequential data sets without FORMAT control. (See figures 2-9 through 2-11.) However, logical records without FORMAT control span buffers whenever the data length exceeds the available buffer space. The spanning of buffers necessitates the green control word (GCW), which will prefix the text of each subrecord of unformatted records. The green control word's format is as follows:

BYTE 1	BYTE 2	BYTE 3	BYTE 4

Byte 1 - contains 0 except for the green word of the last subrecord which contains the number of subrecords (this is used for backspacing).

Bytes 2-4 - contain number of bytes of text in the subrecord.

The green word is considered as part of text when specifying the RECSIZE and BLKSIZE parameters, i.e., RECSIZE=TEXT SIZE+4 bytes.

The user can specify that the green control word not prefix the text of each subrecord for BTAM variable-length records only. This may be done by including the following statement within the FORTRAN source program.

CALL GREEN (DS1, DS2,....., DS1,.....)

where: Each DS1 represents the data set reference number associated with that I/O device which will contain the unformatted logical records without the GCW.

Note: This statement must appear before any other statement that references the I/O devices being used.

Without the green word, the TEXT SIZE will be equal to the RECSIZE. The four-byte segment control word is modified to have the following format:

SCW	LRECL	XX	00
	2 Bytes	2 Bytes	

where: LRECL represents the length of the record segment.

XX represents a code describing the status of the current subrecord in the unformatted record, that is:

XX	STATUS
00	ONLY subrecord in the unformatted record
01	FIRST subrecord in the unformatted record
02	MIDDLE subrecord in the unformatted record
03	LAST subrecord in the unformatted record

Example 1 (see figure 2-9): Assume BLKSIZE=32 and RECSIZE=24

WRITE(18)Q,R

where: Q and R are real 8-byte variables.

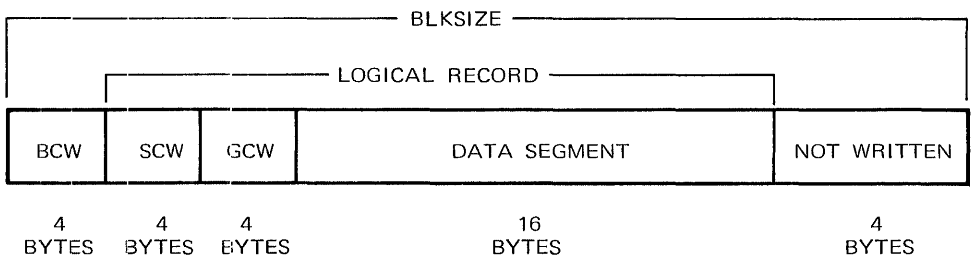


FIGURE 2-9. VARIABLE-LENGTH UNBLOCKED RECORDS, NO FORMAT CONTROL, ONE RECORD SEGMENT

Example 2 (see figure 2-10): Assume BLKSIZE=32 and RECSIZE=24

WRITE(18)Q,R,S,V,X

where: Q,R and V are real 8-byte variables.

S and X are real 4-byte variables.

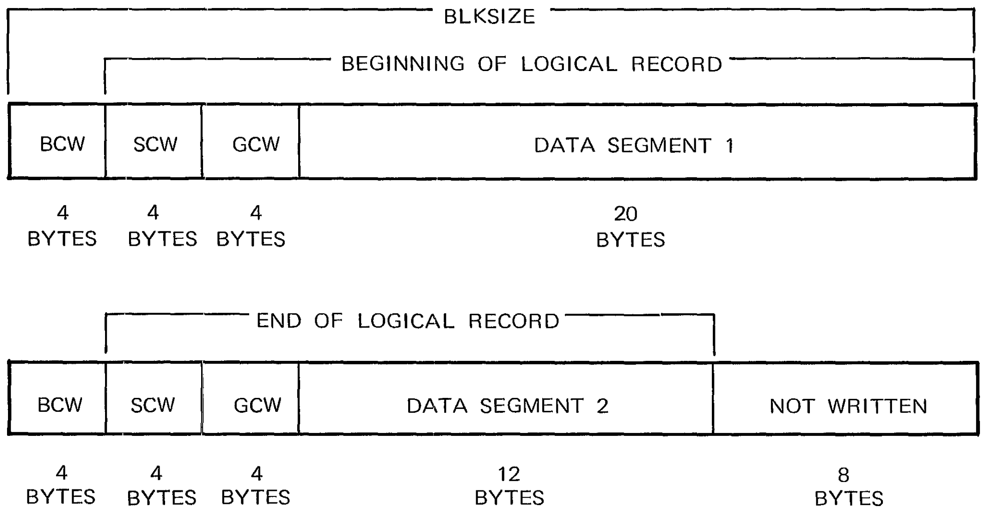


FIGURE 2-10. VARIABLE-LENGTH UNBLOCKED RECORDS, NO FORMAT CONTROL, TWO RECORD SEGMENTS

Example 3 (see figure 2-11): Assume BLKSIZE=36 and RECSIZE=12

WRITE(18)A
.
.
.
WRITE(18)B
.
.
.
WRITE(18)E

where: A is a real 8-byte variable.

B and E are real 4-byte variables.

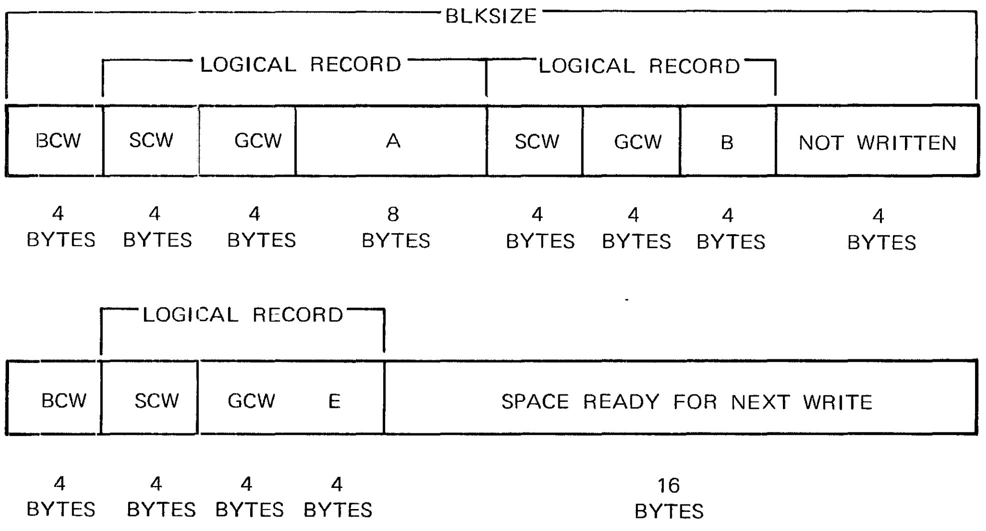


FIGURE 2-11. VARIABLE-LENGTH BLOCKED RECORDS, NO FORMAT CONTROL

Example 4: (See figure 2-12.) Assume BLKSIZE=64 and RECSIZE=56

```
CALL GREEN (33)
.
.
.
.
.
WRITE (33) (ARRAY(I), I=1,30)
```

where: ARRAY is defined to be a real *4 array with dimension 30,

DIRECT-ACCESS DATA SET SPECIFICATION

Direct Access is implemented with the Indexed Sequential Access method (ISAM) of DMS. Therefore, the user must provide space in the record for ISAM keys when he writes his DATAD macro.

Define File Statement

The record length and buffer length for a data set may be specified by the user through the DEFINE FILE Statement. The DEFINE FILE statement and its associated parameters are described below:

DEFINE FILE a(m,r,f,v)

a = data set reference number

m = number of records in the file

r = record size

$f=\left\{\begin{array}{c}E\\U\\L\end{array}\right\}$

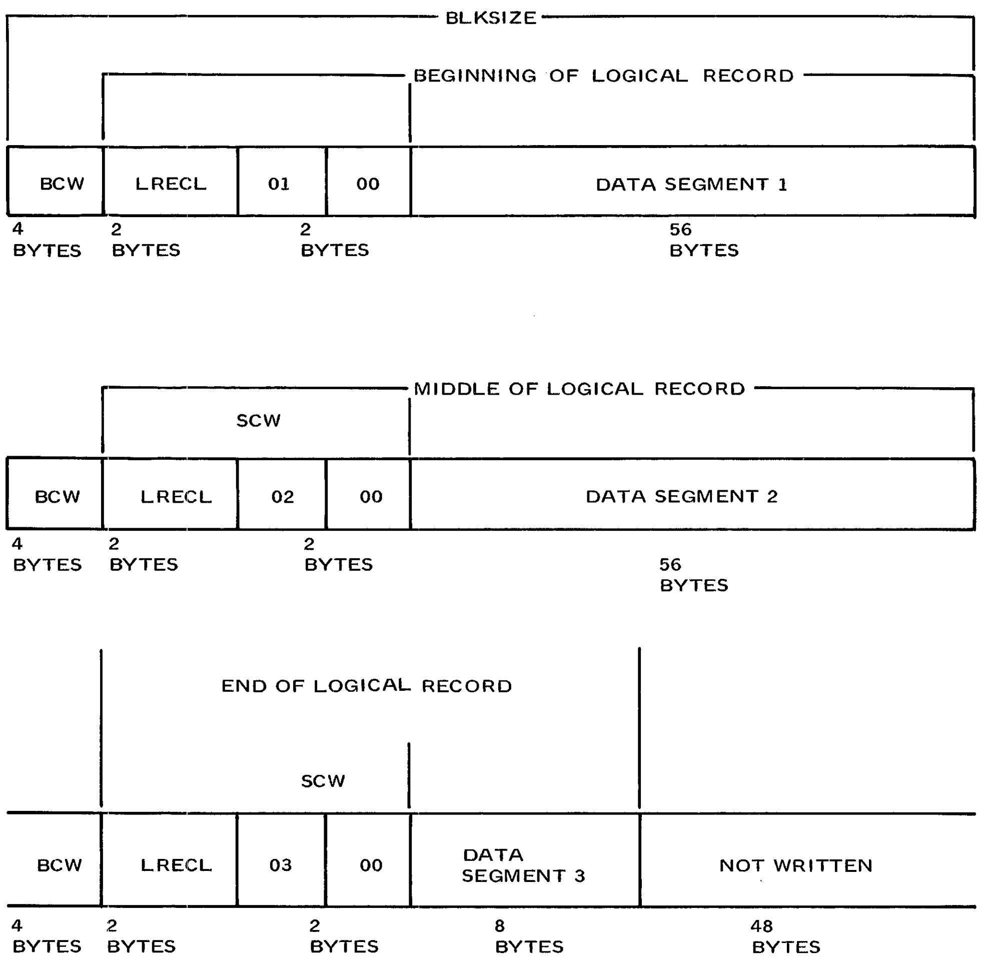


FIGURE 2-12. VARIABLE-LENGTH UNBLOCKED RECORDS, NO FORMAT CONTROL

where: E = access with format statement

U = access W/O format control

L = either E or U

v = associated variable which is utilized by the user to direct the retrieval and writing of specified records. Upon return to the user program the associated variable can be interrogated to obtain the value of the next record number in the file.

Record Formats

The record format for direct access data set is the same as the format described for fixed-length, unblocked records written for sequential data sets.

Records written without FORMAT control on direct-access data sets contain the FORTRAN green control word. The logical record can exceed the record length specified in the DEFINE FILE statement, and if a record segment is shorter than the record length, the remaining portion of the record is padded with spaces (see figure 2-13).

Example: A DEFINE FILE statement has specified the record length for a direct access data set as 24. This statement is then executed. WRITE(9'IX)DP1,DP2,R1,R2

where: DP1 and DP2 are double-precision variables.
R1 and R2 are real variables.
IX is an integer variable that contains the record position.

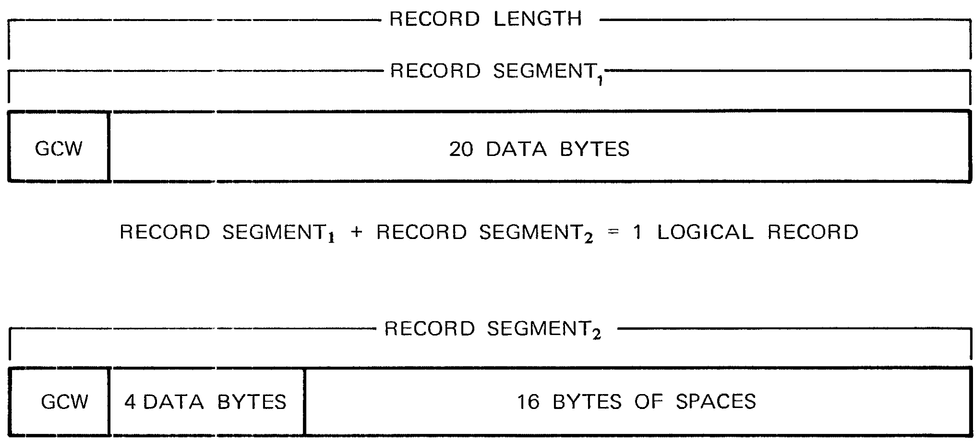


FIGURE 2-13. LOGICAL RECORD (NO FORMAT CONTROL) FOR DIRECT ACCESS

BACKSPACE, END FILE, and REWIND operations are ignored for direct access data sets.

FORTRAN Data Management Library Routines

The FORTRAN Data Management library routines provide the following three Data Management facilities and associated functions:

1. FORTRAN data definition establishment facility.

2. FILE management and catalog facilities.

   a. Object-time data control block generation, validation, and updating.

   b. Dynamic file allocation and catalog default function.

   c. Dynamic buffer management.

d. File positioning and label checking.

e. File purging.

f. File copying.

3. INPUT-OUTPUT facilities.

a. Input-output control.

b. Logical record blocking and deblocking.

c. Access driver.

d. FORMAT controlled input conversions.

e. FORMAT controlled output conversions.

f. NAMELIST controlled input conversions.

g. NAMELIST controlled output conversions.

h. Binary (non-FORMAT, non-NAMELIST) input-output.

i. BACKSPACE, REWIND, and END FILE function.

FORTRAN Data Definition Establishment Facility

STANDARD SYSTEM DATA DEFINITIONS

The library contains a copy of the standard data sets constituting the basic data definitions established for use in RCA time sharing systems. Table 2-1 describes each of these data sets.

TABLE 2-1. VMOS STANDARD FORTRAN DATA SETS

Data Set Number	Device Name	File Type	Action Macro	Recform Type
1	Terminal	SYSDTA	RDATA	VARUNB
2	Terminal, Printer	SYSOUT	WROUT	VARUNB
5	Card Reader	SYSIPT	RDATA	VARUNB
6	Printer	SYSLST	WRLST	VARBLK
7	Card Punch	SYSOPT	WRTOT	VARBLK
97	Card Reader	SYSIPT	RDATA	VARUNB
98	Card Punch	SYSOPT	WRTOT	VARBLK
99	Printer	SYSLST	WRLST	VARBLK

Data sets 7, 97, and 98 have been supplied to support TOS/TDOS compatible system logical files. Also, data set reference numbers 97, 98, and 99 are referenced when the programmer uses the READ (old form), PUNCH, and PRINT statements, respectively.

ALTERNATIVE SYSTEM/CORE DATA DEFINITION

Facilities have been provided which permit the programmer to define additional data sets, redefine any or all of the standard data sets, and designate that virtual memory be used as the data set. Additional data sets must, however, reference the same file types as those assigned to standard data sets (see table 2-1).

In order to implement the specification of any of the alternative data set options, the programmer must assemble both a DDS macro and a DVLST macro. The DDS macro defines the characteristics of the data set. The DVLST macro creates a device list table containing the data set reference numbers of the data sets used by the FORTRAN program.

DDS (Define Data Set) Macro

```
Operation   Operand

DDS         DSREF = data set reference number

            ┌ RECFORM=  ⎧ FIXBLK ⎫ ┐
            │           ⎪ FIXUNB ⎪ │
            │           ⎨ VARUNB ⎬ │
            │           ⎪ VARBLK ⎪ │
            └           ⎩ UNDEF  ⎭ ┘

            [,BLKSIZE=  bytes]

            [,RECSIZE=  bytes]

            ┌ ,TYPEFLE= ⎧ INPUT  ⎫ ┐
            │           ⎪ OUTPUT ⎪ │
            │           ⎨ INOUT  ⎬ │
            │           ⎪ OUTIN  ⎪ │
            └           ⎩ DUMMY  ⎭ ┘

            ┌ ,CONTROL= ⎧ ASA  ⎫ ┐
            │           ⎨ MACH ⎬ │
            └           ⎩ NO   ⎭ ┘

            [,DCBEQU=   dcb-name]

            ┌ [,DEVICE= ⎧ SYSTEM ⎫ ┐
            └           ⎩ CORE   ⎭ ┘

            ┌ ,DEVADDR= ⎧ SYSDTA ⎫ ┐
            │           ⎪ SYSIPT ⎪ │
            │           ⎨ SYSLST ⎬ │
            │           ⎪ SYSOUT ⎪ │
            └           ⎩ SYSOPT ⎭ ┘
```

DSREF=

A two-digit integer supplied by the programmer to refer to a data set.

RECFORM=

FIXBLK	Blocked, fixed-length records.
FIXUNB	One per block, fixed-length records.
VARUNB	One per block, variable-length records with the length specified in the first 4 bytes of the block.
VARBLK	Blocked, variable-length records with the length specified in the first 4 bytes of each record.
UNDEF	Logical record of variable size. No length specification. This is the default case.

BLKSIZE=

The largest block to be read from or written to the data set. It establishes the minimum buffer size.

If no BLKSIZE is specified, the block size and record size (RECSIZE) are associated by the system with the device address (DEVADDR=). The following table specifies this relation:

DEVADDR=	RECSIZE=	BLKSIZE=
SYSDTA	80	88
SYSOUT	73	81
SYSLST	133	141
SYSOPT	80	80

RECSIZE=

The record size for fixed-length records. The maximum size for variable-length and undefined records.

If no RECSIZE is specified, the record size and block size (BLKSIZE) are associated by the system with the device address (DEVADDR=). The following table specifies this relation.

DEVADDR=	RECSIZE=	BLKSIZE=
SYSDTA	80	88
SYSOUT	73	81
SYSLST	133	141
SYSOPT	80	80

TYPEFLE=

<u>INPUT</u>	Indicates input processing. This is the default option.

	OUTPUT	Indicates output processing.
	INOUT	Indicates input processing with the ability to write to the file.
	OUTIN	Indicates output process with the ability to retrieve records from file.
	DUMMY	INPUT-OUTPUT operation is inhibited.
CONTROL=		
	ASA	The first character of the record will control spacing according to the ASA conventions: blank means single space, 0 means double space, 1 means advance to next page, and + means no space.
	MACH	The first character will be given to the printer directly for carriage control.
	NO	The first character will not be used for carriage control. The first character will be printed and lines will be single spaced. This is the default option. Note: If a task is conversational and the output device is the terminal, the control character is ignored.

DCBEQU=

The name of an existing DCB to which the DCB created by the DDS is to be equated.

DEVICE=		
	SYSTEM	One of the system logical files. If this option is used, the DEVADDR parameter must be entered.
	CORE	Virtual memory.
DEVADDR=		
	SYSDTA	Specifies the current data input stream: a terminal, the card reader, or a cataloged direct access file.
	SYSIPT	Specifies the current data input stream: a card reader or cataloged direct access file. SYSIPT supplies Class I program compatibility.
	SYSLST	Specifies the printer for output listings.
	SYSOUT	Specifies the output destination of messages: terminal (conversational task) or printer (nonconversational (task).
	SYSOPT	Specifies that output be sent to the card punch.

DVLST (Device List) Macro

Operation	Operand
DVLST	Data set no_1,..., Data set no K

where: K≤99

Redefine Standard Data Sets

The redefinition of any standard data set requires the use of both the DDS macro and the DVLST macro. The DDS macro is used to describe the data set being redefined. The DVLST macro creates the device list table for all the data sets to be used by the program.

The following procedure describes the optimum procedure for redefining standard data sets.

1. Write and assemble a DDS macro describing the new characteristic of each redefined data set.

2. Write and assemble a DDS macro describing the standard characteristics of each of the remaining standard data sets to be used by the program. Although the characteristics of these data sets are not being altered, they must be defined. The characteristics of the standard data sets are given in table 2-2.

3. Write and assemble a DVLST macro listing the numbers of all the data sets (redefined and standard) to be used by the program.

Example:

The programmer is redefining data set 6, SYSLST, and also wishes to use data sets 1 and 2, SYSDTA and SYSOUT, in their standard form.

1. DDS macro for redefinition of data set 6 to change the data set number to 10 and record size to 120.

```
DDS        DSREF = 10,RECFORM=VARUNB,
           BLKSIZE=141,RECSIZE=120,TYPEFLE=OUTPUT,
           CONTROL=NO,DEVICE=SYSTEM,DEVADDR=SYSLST
```

2. DDS macro for reestablishment of the default (standard) attributes of data set 1.

```
DDS       DSREF=1,RECFORM=VARUNB,BLKSIZE=88,RECSIDE=
          80,TYPEFLE=INPUT,DEVICE=SYSTEM,DEVADDR=SYSDTA
```

Note: The CONTROL= and DCBEQU= parameters are not applicable to the standard definition of data set 1.

3. DDS macro for reestablishment of the default (standard) attributes of data set 2.

```
DDS       DSREF=2,RECFORM=VARUNB,BLKSIZE=81,RECSIZE=73,
          TYPEFLE=OUTPUT,CONTROL=NO,DEVICE=SYSTEM,
          DEVADDR=SYSOUT
```

Note: The DCBEQU= parameter is not applicable to the standard definition of data set 2.

4. DVLST macro to create the Device List Table to be used by the program.

```
DVLST 1,2,10
```

Note: Only those data sets for which a DDS was written must be referenced in the DVLST.

Add New System Data Set

The addition of a new system data set to the Device List Table of standard data sets requires the use of both the DDS and DVLST macros. The DDS macro is used to describe the data set being added to the Device List Table. The DVLST macro updates the Device List Table to include the new data set reference number.

The following describes the procedure for adding a new system data set to the Device List Table.

1. Write and assemble a DDS macro for each of the data sets being added to the Device List Table.

2. Write and assemble a DVLST macro listing both the new data set and all of the standard data sets.

Example:

The programmer wishes to add a third SYSLST data set to the Device List Table. The data set reference number is to be 12.

1. DDS macro describing the characteristics of data set 10.

```
DDS       DSREF=12,RECFORM=VARUNB,BLKSIZE=141,
          RECSIZE=133,TYPEFILE=OUTPUT,CONTROL=NO,
          DCBEQU=DS99,DEVICE=SYSTEM,DEVADDR=SYSLST.
```

2. DVLST listing all standard data sets including the one being added to the Device List Table.

```
DVLST     1,2,5,6,7,97,98,99,12
```

To designate that a portion of virtual memory be used as a data set, the programmer must use both the DDS macro and the DVLST macro. The DDS macro defines the characteristics that the programmer wishes to assign to virtual memory. The DVLST macro creates the proper entry in the Device List Table.

The following describes the procedure for designating that virtual memory be used as a data set:

1. Write and assemble a DDS macro describing the characteristics that virtual memory is to assume as a data set.

2. Write and assemble a DVLST macro listing all the standard data sets and the data set number to be assigned to virtual memory.

Example:

The programmer wishes to designate that virtual memory be used as a data set. The data set reference number if to be 8.

1. DDS macro defining the characteristics of virtual memory.

```
DDS        DSREF=8,RECFORM=VARUNB,BLKSIZE=88,
           RECSIZE=80,TYPEFLE=INOUT,DEVICE=CORE
```

Note: The CONTROL=, DCBEQU=, and DEVADDR= parameters refer only to system logical files and their corresponding devices.

2. DVLST macro to add data set 8 to the Device List Table.

```
DVLST      1,2,5,6,7,97,98,99,8
```

TABLE 2-2. VMOS STANDARD DATA SET CHARACTERS

DSREF=	1	2	5	6	7	97	98	99
RECFORM=	VARUNB	VARUNB	VARUNB	VARUNB	FIXUNB	VARUNB	FIXUNB	VARUNB
BLKSIZE=	88	81	88	141	80	88	80	141
RECSIZE=	80	73	80	133	80	80	80	133
TYPEFLE=	INPUT	OUTPUT	INPUT	OUTPUT	INPUT	INPUT	OUTPUT	OUTPUT
CONTROL=	N/A	NO	N/A	NO	N/A	N/A	N/A	NO
DCBEQU=	N/A	N/A	DS97	DS99	DS98	N/A	N/A	N/A
DEVICE=	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM	SYSTEM
DEVADDR=	SYSDTA	SYSOUT	SYSIPT	SYSLST	SYSOPT	SYSIPT	SYSOPT	SYSLST

FORTRAN programmers may also define data sets for device address references other than SYSTEM or CORE. The DCB's (Data Control Blocks) established for these additional data sets may be ISAM, BTAM, or EAM.

The programmer establishes the ISAM or BTAM DCB by using the FILE command with the LINK parameter referencing the data set number. When initial access is made to the data set, the data set reference number will be added to the Device List Table. Subsequent accesses to the data set number will reference the Device List Table.

If the programmer accesses a nonstandard data set and has not issued a FILE command describing the data set, the system assigns an EAM DCB for that data set. EAM data sets are system defaults; users cannot specify their own EAM data sets.

FILE Command Parameters for ISAM DCB

The FILE command parameters listed in this section are relevant to FORTRAN processing for ISAM DCB's. The parameters shown are only pertinent to FORTRAN direct-access files. The values explicitly specified are the default values peculiar to FORTRAN and are not necessarily those used by the Data Management System (DMS). For alternative values for these parameters, consult the Data Management System Reference Manual.

DMS processing requires that the OPEN type for a new file must be OUTPUT or OUTIN. For an old file, the OPEN type must be INPUT or INOUT. The programmer must issue the appropriate FILE command for the type of file to be processed.

The FILE command parameters for an ISAM DCB follow:

Operation	Operand
FILE	filename, LINK=DSETnn, FCBTYPE=ISAM,RECSIZE=n [,BLKSIZE=STD] [,KEYLEN=4] [,DUPKEY=NO] [,KEYPOS=n] [,RECFORM=n] [OPEN=type]
filename	The name of the file. This entry is required.
LINK=DSETnn	nn is the two-digit reference number of the data set. This entry is required.
FCBTYPE=ISAM	This entry is required and must be specified as shown.
RECSIZE=n	n is the length in bytes of the logical records in the file. If RECFORM=V is specified, n cannot exceed 2040 bytes. If RECFORM=F is specified, n cannot exceed 2048 byte. This entry (RECSIZE=) is required.

BLKSIZE=STD	This entry is not required. If specified BLKSIZE must be STD.
KEYLEN=4	This entry is not required. A keylength of 4 will be assumed by FORTRAN if the KEYLEN parameter is not specified. If KEYLEN is specified, it must be 4.
DUPKEY=NO	This entry is not required. If the DUPKEY parameter is not entered, no key duplicating will be allowed.
KEYPOS=5 =1	This entry is not required. If RECFORM=V is specified, the record key is assumed to begin in byte 5 of the record. If RECFORM=F is specified, the record key is assumed to begin in byte 1 of the record.
RECFORM=F =V	This entry is not required. RECFORM=F specifies fixed form records; RECFORM=V specifies variable form records. If the RECFORM parameter is not specified the record form is assumed to be variable (V).
OPEN=INPUT =OUTPUT =OUTIN =INOUT	This entry is not required. If no file opening specification is entered, OPEN=OUTIN is assumed.

Example:

The following shows the use of the FILE command to specify an ISAM data set. The reference number is 25 and the record size is 24.

```
FILE TJ.FOR,LINK=DSET25,FCBTYPE=ISAM,RECSIZE=24
```

The preceding example lists only the required parameters. All others were allowed to assume their default values.

FILE Command Parameters for BTAM DCB

The FILE command parameters listed in this section are relevant to FORTRAN processing of BTAM DCB's. The parameters shown are the minimum set pertinent to FORTRAN tape files. For a description of the remaining FILE command parameters for BTAM files, consult the Data Management System Reference Manual.

The minimum FILE command parameters for a BTAM DCB follow:

Operation	Operand
FILE	filename, LINK=DSETnn, FCBTYPE=BTAM, RECSIZE=n, BLKSIZE=n, RECFORM=$\left\{\begin{array}{c}\text{F}\\\text{V}\\\text{U}\end{array}\right\}$

filename	The name of the file. This entry is required.
LINK=DSETnn	nn is the two-digit reference number of the data set. This entry is required.
FCBTYPE=BTAM	This entry is required and must be specified as shown.
RECSIZE=	n is the length in bytes of a logical record. The maximum record length is 4096 bytes. See Program Considerations for the use of this operand.
BLKSIZE=	n is the size in bytes of a block of logical records. See Programming Considerations for the use of this operand.
RECFORM=F =V =U	F = fixed length records. V = variable-length records. This is the default value. U = undefined records. See Programming Considerations for the use of this operand.

Programming Considerations

The programmer must specify a value for either RECSIZE or BLKSIZE when issuing the FILE command. Table 2-3 describes the use of these operands in conjunction with the record form (RECFORM) specifications for BTAM record blocking.

TABLE 2-3. SPECIFICATIONS FOR BTAM RECORD BLOCKING

Blocking Type	RECFORM	RECSIZE	BLKSIZE	Comment
FIXUNB	F	Yes	Yes	
FIXBLK	F	Yes	Yes	BLKSIZE must be K (RECSIZE). Where K is an integer.
VARUNB	V	No	Yes	If only BLKSIZE is given, BLKSIZE=RECSIZE +8 bytes.
VARBLK	V	Yes	Yes	BLKSIZE must be greater than RECSIZE +8 bytes.
UNDEF	U	Yes	Yes	If both are given, the system will set both BLKSIZE and RECSIZE= 4096 bytes. If only RECSIZE is given, the system will set BLKSIZE equal to maximum RECSIZE. If BLKSIZE is given, the system will set RECSIZE equal to BLKSIZE.

Example:

The following FILE command defines the characteristics of a BTAM data set whose reference number has been specified as 25.

```
FILE        TJ.FOR, LINK=DSET25, FCBTYPE=BTAM, DEVICE=T9N,
            RECFORM=F, RECSIZE=256, BLKSIZE=1024
            VOLUME=AA1234
```

In the preceding command specification, the DEVICE and VOLUME parameters were not required by FORTRAN for BTAM processing. However, the DEVICE and VOLUME parameters should be specified in order to assure proper processing by the operating system.

As both RECSIZE and BLKSIZE were given and RECFORM was specified to be F, the blocking type is fixed block (FIXBLK).

If the programmer does not specify a particular type labeling convention, standard labels will be supplied by the system.

EAM DCB Characteristics

If a data set reference number other than 1,2,5,6,7,97,98, or 99 has been used and a file command has not been issued with LINE=DSETNN before execution, the user will automatically get an EAM file.

EAM files are not cataloged and are temporary.

EAM files cannot be shared or multiply opened.

EAM files always reside on public volumes, and user storage allocation, device assignment, or label processing is not permitted.

An EAM file may only be used for files with sequential organization and access. No record keys are permitted.

EAM DCB's conform to the following specifications

RECFORM = VARBLK

RECSIZE = 2040

BLKSIZE = 2048

File Management and Cataloging Facilities

FORTRAN will make full use of the DMS FILE, CATALOG, and FCB macros, to provide the following functions:

File positioning and label checking.

File purging.

File copy.

For a full description of these macros, refer to the Data Management System Reference Manual.

Mathematical Routines

All the TOS/TDOS FORTRAN mathematical routines are supported and are reentrant.

Debugging Routine

The function of the debugging routine is to give the programmer a report on the cause of an abnormal termination and the status of the subprogram linkage at the time of termination. In the case of an I/O error termination, it gives pertinent information about the I/O statement and the data involved.

The user specifies

```
/PARAM DEBUG=YES
```

before compilation, the DEBUG subroutine also displays the line number of the source statement where the error occurred.

The object-time error messages are directed to the task's SYSOUT file. For a conversational task, this is the user's terminal; for a nonconversational task, the printer.

In addition, if the user specifies:

```
/PARAM SYMDIC=YES
```

the FORTRAN Compiler generates a symbol dictionary so that the user may employ symbolic IDA at object-program execution time.

3. ASSEMBLY

INTRODUCTION

The Assembler is a nonconversational Class II (pageable) program which runs under control of the Spectra-70 Time Sharing Operating System (TSOS). It generates object modules that can be executed under TSOS as either Class I (non-pageable) or Class II (pageable) programs.

The Assembler is a disc oriented program which generates object modules on EAM direct access files. The source program correction facility of the Assembler also handles BTAM magnetic tape files as input and output.

The following are some of the facilities provided by the Assembler:

1. Source input can be retrieved from a cataloged file, from a card deck, from a command stream, or from a remote terminal. The source program correction facility accepts BTAM magnetic tape input.

2. The generated object module is written on the EAM object module file and can be punched on cards before task termination. The source correction facility writes its output to a BTAM private tape file.

3. Requested listings are written on EAM files and automatically spooled out at task termination.

4. An Internal Symbol Dictionary (ISD) can be generated, to be used by the Interactive Debugging Aid (IDA) for symbolic debugging at program execution time.

5. A diagnostic file can be generated on disc for query through the Post-Assembly Diagnostic Routine (ADIAG). This file also acts as a saved file of the assembled listings. The ADIAG routine is used to print the listings when desired.

The Assembler operates nonconversationally, that is the user cannot direct the process of assembling once it has begun. However, there can be dynamic communication between the Assembler and the programmer in the sense that the programmer enters source input from a remote terminal.

After assembly, the conversational programmer may invoke the Post-Assembly Diagnostic Routine, ADIAG, to interrogate the diagnostic file. In this way, the programmer can obtain immediate information about the assembly, such as the number of errors noted by the Assembler and a description of any such errors. This information enables the programmer to decide whether to correct the source program and recompile or whether to execute the object program.

At object program execution time, the programmer can employ the IDA (Interactive Debugging Aid) System to help him find logic errors in the object program. If an Internal Symbol Dictionary (ISD) was created at assembly time the IDA commands can refer to tags by their symbolic names; without an ISD, a tag must be referred to by its hexadecimal address.

EQUIPMENT CONFIGURATION

The Assembler requires approximately 80,000 bytes of virtual memory. The following minimum TSOS system is required for Assembly:

1–Processor (262 kb)
2–Disc Storage Units
1–Paging Drum

In addition, the system must include a Communications Controller Multichannel (CCM) with associated buffers and if it is to support conversational users, remote terminals.

Optional Equipment

2–Tape Drives - if the source correction facility is used.
1–Card Reader - if input is to come from cards.
1–card Punch - if the object module is to be punched into cards.
1–Printer - if listings are required.

ASSEMBLY LANGUAGE

The source language processed by the Assembler is described in the Assembly System Reference Manual. The macros which an assembly language programmer can use to communicate with the Executive and the Data Management System (and which are contained in the standard macro library) are described in the Executive Macros Reference Manual, and the Data Management System Reference Manual, respectively.

Input

The Assembler reads its input by using the RDATA macro which instructs the Executive to read a record from the current task's SYSDTA file. The SYSDTA file may be equated to a cataloged file, the card reader, a remote terminal, or the command stream (See figure 3-1).

When the assembly source correction facility is used, the input must be a magnetic tape file which is accessed by the Basic Tape Access Method (BTAM).

Disc-Resident Source Files

An Assembly language source file which resides on disc must be either a Sequential Access Method (SAM) file or an Indexed Sequential Access Method (ISAM) file. In the latter case, the 8-byte numeric keys must be situated in the first 8 data bytes of the record. The eight-byte keys are printed in columns 73-80 of the Assembly source listing. ISAM files with this construction may be directly created, updated, and edited by the File Editor (see the File Editor Reference Manual). The File Editor can also edit SAM files indirectly.

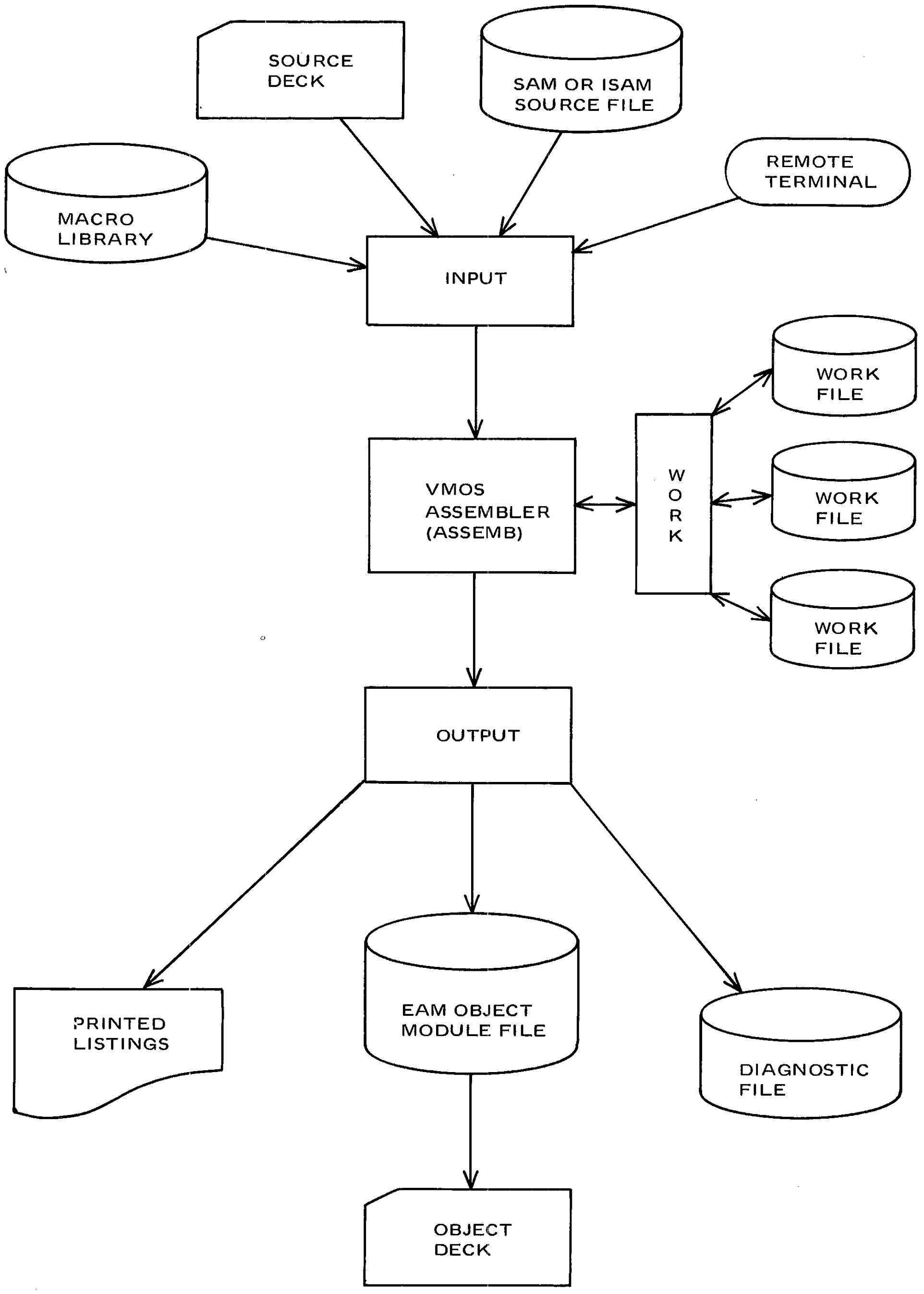


FIGURE 3-1. ASSEMBLER INPUT-OUTPUT FLOWCHART.

If the Assembler is to retrieve its input from a file, it is the user's responsibility to ensure that the source input is contained in a cataloged file and to use the Executive command SYSFILE to direct SYSDTA to the source file. (See examples at the end of this section.)

Note: If the user wishes to do any additional processing after the end of the compilation, for example, the Library Maintenance Routine, he must be careful at that time to redirect SYSDTA from his source file to whatever source is required by the subsequent routines. For instance, a conversational user wishing to redirect SYSDTA to his terminal after compilation would issue the following command:

```
/SYSFILE SYSDTA=(SYSCMD)
```

Card Input

A programmer may wish to enter the entire assembly as a background task in the form of a card deck containing a /LOGON card, cards containing commands to the Assembler, the source program, and a /LOGOFF card. The operator places the deck in the card reader and initiates execution of the task by entering the RCARD command on the console. The entire card deck is then spooled in and executed. In this case, the programmer should direct SYSDTA so that the Assembler can take its input from the spooled in file, thus:

```
/SYSFILE SYSDTA=(SYSCMD)
```

In a conversational task, SYSDTA is, by definition, the programmer's remote terminal. A conversational programmer initiating an assembly from the terminal might wish to instruct the Assembler to take its input from a source deck which the programmer had previously requested the operator to place in the card reader. In this case, the programmer should redirect SYSDTA from the terminal to the card reader by giving the command:

```
/SYSFILE SYSDTA=(CARD)
```

The operator receives a message asking him to confirm that the deck is actually in the card reader. After the deck has been read, the programmer should direct SYSDTA from the card reader in order to free it for use by the system.

Terminal Input

The conversational programmer may enter programs directly from a remote terminal. In this case, the programmer must be sure that SYSDTA is directed to the terminal, redirecting it there, if necessary, by the command

```
/SYSFILE SYSDTA=(SYSCMD)
```

before starting an assembly.

The Assembler types an asterisk (*) on the terminal, indicating that it is ready to receive source input. The programmer types a line of source code, not longer than 80 characters and conforming to the usual margin conventions (i.e., columns 1, 71 and 16 are used as the beginning, ending and continuation columns, respectively). The Assembler processes this line, then types another asterisk (*) to signify that it is ready for another input line. This process goes on until the programmer has entered the entire source program; the assembly language verb END indicates the end of a CSECT to the Assembler. The Assembler then takes control and processes the preceding CSECT. To terminate execution of the Assembler when SYSDTA is directed to the terminal, the programmer should press the ESCAPE key and, when the system responds with a slash, /, type the command EOF. The system responds with another slash and the programmer may enter any other system command.

Macro Libraries

A collection of macro definitions can be made available to more than one source program by placing the macro definitions in a macro library. If the programmer includes a macro instruction call in the source program the Assembler takes the corresponding macro definition from the macro library and places it in the output object module and in the source listing. This process is known as expansion. The Assembler accesses two types of macro libraries, the system macro library (MACROLIB) and a private macro library (MACROALT) which an individual programmer has created for his own use. Both the system macro library and a private library may be referenced in the same assembly. It is the programmer's responsibility to ensure that only Class 1 macros are used in Class 1 programs and Class 2 macros in Class 2 programs.

The system macro library is supplied with the system and is available to all programmers. Macros on the system library may call other macros on the system library but may not refer to macros in a programmer's private library unless these macros have been used in the source program. If however, the source program refers to macros in the private library, then the macros in the system library may refer to those private macros.

An individual programmer must create his private library through the use of the Macro Library Update routine (MLU). Macros in a private library may refer to macros in the same library or in the system macro library. If a source program refers to a macro which appears in both the system library and the programmer's private library, the Assembler takes the macro expansion from the private library.

If the programmer wishes the Assembler to refer to a private macro library, the following commands must be issued before executing the assembly:

/PARAM ALTLIB=YES   which indicates that he intends to use a private library.

/FILE Macro-library-file-name,LINK=ALTLIB   which supplies the name of the macro library file. The LINK=ALTLIB parameter indicates that the named file is to serve as a private macro library.

If the private macro library is given the file name MACROALT, the FILE command is not necessary. (Caution: The Assembler always tries to open a file named MACROALT as a private library. Therefore, if there is any possibility that a programmer may ever execute the Assembler, he should not use the filename MACROALT for any purpose other than as a private macro library. The mere presence of a file named MACROALT causes the Assembler to try to use it to expand macros.

If, in a particular assembly, the programmer does not wish the Assembler to search the MACROALT file, the programmer can suppress this search by issuing the command:

```
/FILE * DUMMY, LINK=ALTLIB
```

before invoking the Assembler.

If a programmer wishes to substitute a private macro library for the system library in a particular assembly, the programmer can do so by issuing the command:

```
/FILE macro-library-file-name,LINK=SYSLIB
```

before invoking the Assembler.

Source Input Correction

Assembly programs may be introduced through the source facility from SYSDTA or from a private BTAM magnetic tape file.

The source program correction facility is described in the TSOS Utility Routines Reference Manaul.

Use of SYSFILE Command

The Executive command SYSFILE allows the programmer to direct SYSDTA to a cataloged data file, the system card reader, or to redirect SYSDTA to its primary assignment or command stream.

```
SYSFILE    SYSDTA=  ⎧filename ⎫
                    ⎨(CARD)   ⎬
                    ⎨(PRIMARY)⎬
                    ⎩(SYSCMD) ⎭
```

filename

Identifies the name of the cataloged source file. The command is invalid if the file is not cataloged.

(CARD)

Identifies the system card reader.

(PRIMARY)

Identifies the primary assignment of the SYSDTA file (the user's terminal in a conversational task, the card reader or the spooled in file in a nonconversational task).

(SYSCMD)

The same source as commands.

Output Object Module

The TSOS Assembler generates an EAM object module file on direct access public storage.

Disc-Resident Object Module

The Assembler automatically writes the object module on an EAM file unless the programmer specifically inhibits this by means of the following command:

/PARAM DISC=NO

Because files created by EAM are temporary (at most, they exist for the duration of the task in which they are created), the programmer who wishes to keep an object module in a permanent disc file should do so by invoking the Library Maintenance Routine (for a description of this routine, refer to the Utility Routines Reference Manual.)

Object Module Card Decks

If the programmer wishes to receive an object module in the form of a deck of punched cards, either of the following commands may be issued:

/PARAM CARD=YES

This command must be issued prior to assembly and is an instruction to the Assembler to punch out the object deck.

/PUNCH *

This command must be issued following the assembly and prior to logoff. When this command is to be used, the programmer must issue the command /PARAM DISC=YES prior to assembly.

Furthermore, once the PUNCH command has been issued, the EAM object module file will no longer be available to the user. If he wishes to save the object module in a library, he should do so by using the Library Maintenance Routine before issuing the PUNCH command.

Class I and II Object Modules

The Assembler can produce either a Class I or a Class II object module.

The programmer should specify the type of object module required in the /PARAM command, as follows:

/PARAM CLASS=1	For Class I object modules
/PARAM CLASS=2	For Class II object modules (default option)

Listings

The Assembler can produce a source code listing and a cross-reference listing. The source code listing, which also shows the generated object code at the left hand side of the list is automatically generated and spooled out at task termination. If the programmer does not require this listing, it may be suppressed by issuing the command:

```
/PARAM ASMLST=NO
```

before executing the assembly.

The cross reference listing is not generated unless the programmer specifically requests it with the command:

```
/PARAM XREF=YES
```

Note: If a conversational programmer requests a diagnostic file which is to be used in the ADIAG routine, a printed listing may be superfluous because the same information can be displayed at the terminal through the ADIAG commands PRINT or END listing.

Diagnostic File

If the programmer issues the command:

```
/PARAM ERRFIL=YES
```

before executing the assembly, the Assembler produces an ISAM diagnostic file which the programmer can query, from the remote terminal by using the ADIAG routine, to ascertain the success of the assembly.

The diagnostic file contains: the External Symbol Dictionary (ESD) listing, the assembly listing proper, the cross-reference listing, error data, and a summary of errors. The diagnostic file is built sequentially as an ISAM file, so that it can be accessed by the Assembler Diagnostic Program.

The Assembler forms the name of the diagnostic file by suffixing the name of the program's first CSECT with the characters .ASSM.DIAG. Thus if the first CSECT (or the START statement) of a program is named ZZZ, the diagnostic file for that program has the name ZZZ.ASM.DIAG. This means that two programs having the same first CSECT name will also have the same diagnostic file name. It should be noted that the diagnostic file is erased and recataloged at the beginning of each assembly. Assume for the two program examples discussed above that the first assembly produces a diagnostic file which is to be examined after the second program is assembled. It is then necessary to change the name of the first program's diagnostic file to avoid its destruction. See the Data Management System Manual for a description of the use of the CATALOG command to change the name of a file.

If the first CSECT of the program is unnamed, the Assembler gives the name ASSM.DIAG to the Diagnostic file.

If the programmer wishes to supply a file name for the Diagnostic file, rather than having the Assembler supplied name, the following command must be issued before invoking the Assembler:

/FILE filename,LINK=DIAGFILE,SPACE=(primary allocation, secondary allocation)

The programmer must also issue another FILE command for this file before invoking the Assembler Diagnostic Routine ADIAG, in this form:

/FILE filename,LINK=DIAGLINK

Intermediate Work Files

The Assembler uses disc resident files, not tapes, for intermediate work space. The Assembler accesses these files by the EAM Access Method of the Data Management System.

USE OF PARAM COMMAND FOR ASSEMBLY

The following options of the Executive command PARAM may be used to direct the execution of the Assembler. The default cases, which are underlined, hold if the user does not enter a particular parameter.

PARAM Command Parameters

ALTLIB =

	YES	A private macro library is to be used during assembly as well as the system macro library.
	<u>NO</u>	Only the system macro library is to be used.
		Note: The ALTLIB parameter is not currently supported.

ASMLST =

	<u>YES</u>	An assembly listing is to be written on a temporary disc file and printed at task termination.
	NO	No listing is to be produced.

CARD =

	YES	The object module is to be written on an EAM (temporary) file (same as DISC= YES. However, CARD=YES will cause the Asembler to automatically have the EAM file punched into cards.
	<u>NO</u>	Specifies that the object module is not to be written to disc unless the option DISC=YES is in effect. If DISC=YES has been specified, the programmer may issue a /PUNCH * command after assembly to have the EAM file punched into a card deck.

DISC =		
	YES	The object module is to be written on the task's object module file (an EAM temporary file named*).
	NO	The object module is not to be written on the EAM file unless CARD=YES is specified.
ERRFIL =		
	YES	All assembly diagnostic information is to be written on an ISAM file for use in the ADIAG routine.
	NO	The diagnostic file is not to be written.
INPUT =		
	SYSDTA	All source language input will originate from SYSDTA.
	filename	Only the source correction input will originate from SYSDTA; the source program is to be retrieved from the specified BTAM file. The file name must be fully qualified.
OUTPUT =		
	ASMSRCE	The output of the source correction facility is to be written to the BTAM file named ASMSRCE.
	filename	The output of the source correction facility is to be written on the specified file. The file must be a private BTAM file whose name exists in the TSOS catalog.
SYMDIC =		
	YES	An Internal Symbol Dictionary is to be constructed with entries for all tags, including EQU, ENTRY and EXTRN names.
	NO	NO ISD is to be created.
XREF =		
	YES	A symbol table map and cross-reference listing is to be written on a temporary file and printed at task termination.
	NO	No cross-reference listing is to be produced.

Example Of a Conversational Session Where a Programmer Assembles a Program and Stores That Object Module In An Object Module Library

```
(1) /PARAM ERRFIL=YES,ASMLST=NO
(2) /SYSFILE SYSDTA=DUM
(3) /EXEC ASSEMB
    %L001 PROGRAM LOADING
    VERS. 0009 OF TSOS ASSEMBLER READY
(4) FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTDS
(5) /SYSFILE SYSDTA=(PRIMARY)
(6) /FILE OMLIB,SPACE=(12,3)
    /FSTAT OMLIB
    %0000024 OMLIB
(7) /EXEC LMR
    %L001 PROGRAM LOADING
     LMR VER0A4 100170 READY.ENTER CONTROLS.
(8) *CONTROL OUTFILE=OMLIB
(9) *SOURCE *
(10) *ADD OBJMOD=TEST
(11) *END
     %D U600 LMR NORMAL HALT.
    /LOGOFF
    %C E420 LOGOFF AT 1638 ON 12/04/70, FOR TSN 7585.
    %C E421 CPU TIME USED: 0083.3315 SECONDS.
```

Notes on Above Session:

(1)	User specifies parameters, he wishes to have a diagnostic file created and the source listing suppressed.
(2)	Directs SYSDTA to the file (DUM) containing the Assembly source program.
(3)	Invokes the Assembler.
(4)	Assembler message indicates no errors, therefore user does not invoke ADIAG to scan diagnostic file.
(5)	Re-directs SYSDTA to the terminal.
(6)	Catalogs a file (OMLIB) for the object module library and allocates space for it.
(7)	Invokes Library Maintenance Routine.
(8)	LMR Control statement indicates name of OML file.
(9)	Source * indicates that the object module is in the task's temporary (EAM) object module file.

(10) Supplies name (TEST) of object module to be included in OML.

(11) End of LMR input.

Example of A Conversational Session Where The Programmer Enters Source Statements From A Terminal

```
   %C E222 PLEASE LOGON.
(1) /LOGON USERID, ACCOUNT, PASSWORD
   %C E223 LOGON ACCEPTED AT 1462 ON 12/15/70, TSN 9596 ASSIGNED
   /FSTAT
   %0000006 DUM
   %0000003 EK.COBSRC
   %0000003 CANDID
   %0000006 KREFFT
   %TOTAL PUBLIC PAGES ALLOCATED = 00000018
   /EXEC (EDIT)
   %P001 - DLL V-02
    VERS. 11A OF FILE EDITOR READY
   *O DUM
    OPENED DUM AS OLD V-TYPE FILE.
   *P 100 400
   TEST        START
               BALR  2,0
               USING *,2
               STXIT ,PROGCHEK,,OPINT,URERR
   *P $
               END
   *H
(2) /EXEC ASSEMB
   %L001 PROGRAM LOADING
    VERS. 0009 OF TSOS ASSEMBLER READY
   *TEST       START
   *           BALR  2,0
   *           USING *.2
   *           STXIT ,PROGCHEK,,OPINT,URERR
   *           END
(3) FLAGS IN 00003 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
   *
   /EOF
   /*
   /LOGOFF
   %C E420 LOGOFF AT 1648 ON 12/15/70, FOR TSN 9596.
   %C E421 CPU TIME USED: 0010.2206 SECONDS.
```

Notes on above session:

(1) User logs on.

(2) Invokes the Assembler. No SYSFILE command is required in this case because SYSDTA is automatically directed to the terminal in a conversational session and has not previously been directed elsewhere. The user enters a program consisting of five statements.

(3) The Assembly statement END indicates the end of a CSECT. The Assembler processes this CSECT and then displays an asterisk to allow the user to enter another CSECT if he wishes. If he chooses to end execution of the Assembler he should press the ESCAPE key on his terminal.

(4) After the user presses ESCAPE, the system displays a slash, indicating that the user may enter a system command. He enters EOF to signal end-of-file on SYSDTA, thus terminating the execution of the Assembler.

4. POST-SOURCE LANGUAGE TRANSLATION ROUTINES

INTRODUCTION

Because a programmer writes a problem program to perform a specific function, his work is not complete until his program is actually running without errors and doing the job it was intended to do.

The output of a successful language translation is an object module which is free of language errors. This is only the first step, albeit an important one, in getting a program to fulfill its purpose. After that, the programmer needs to know how to instruct the system to create and execute a loadable program.

Figure 4-1 illustrates the various paths a programmer may take after executing a successful compilation. The various routines which are shown in the figure are described briefly in this chapter.

POST-COMPILATION DIAGNOSTIC ROUTINE

The conversational user may call on the post-compilation diagnostic routine, BDIAG, (see Chapter 5 of this manual) to check the diagnostics, if any, that the compiler produced.

The background compilers generate a permanent diagnostic file on disc if the user so requests at compilation time. The file contains English language error messages describing each source language error detected during compilation and a summary of the severity of errors. The generation of a diagnostic file is independent of the generation of a diagnostic listing even though both contain the same messages. The programmer must specifically request each option through its particular compiliation-time parameter.

After compilation, the conversational programmer at a remote terminal can invoke the BDIAG routine to access the diagnostic file and display the error information he requests.

Diagnostic File Contents

The diagnostic file contains two types of records:

1. Summary record
2. Diagnostic detail record

The file contains one summary record and a variable number of detail records. If the compiler did not generate any diagnostic messages, the file would contain only a summary record indicating that no errors were detected during compilation.

Diagnostic File Compilation-Time Parameter

To obtain a diagnostic file on disc, the programmers using the FORTRAN or COBOL compiler must give the command:

```
/PARAM ERRFIL=YES
```

Cataloging, Allocating, and Naming a Diagnostic File

The user may, if he wishes, create a catalog entry and allocate storage for the diagnostic file by issuing the following DMS command:

```
/FILE    filename,LINK=ERRFIL,SPACE=(Primaryinteger,secondaryinteger)
```

For a full description of the FILE command, see the DMS Reference Manual.

The filename parameter can be the compiler-constructed name or the user assigned name. If the programmer does not catalog and allocate disc space for the diagnostic file, the compiler will do so.

The compilers construct a filename for the diagnostic file by taking the program name (not more than the first eight characters) and prefixing it to the base-name .ERRFIL thus:

Source-program-name.ERRFIL

For a complete description of the post-compilation diagnostic routine, BDIAG, by which the programmer can interrogate the diagnostic file from the terminal, see Chapter 5 of this manual.

POST-ASSEMBLY DIAGNOSTIC ROUTINE

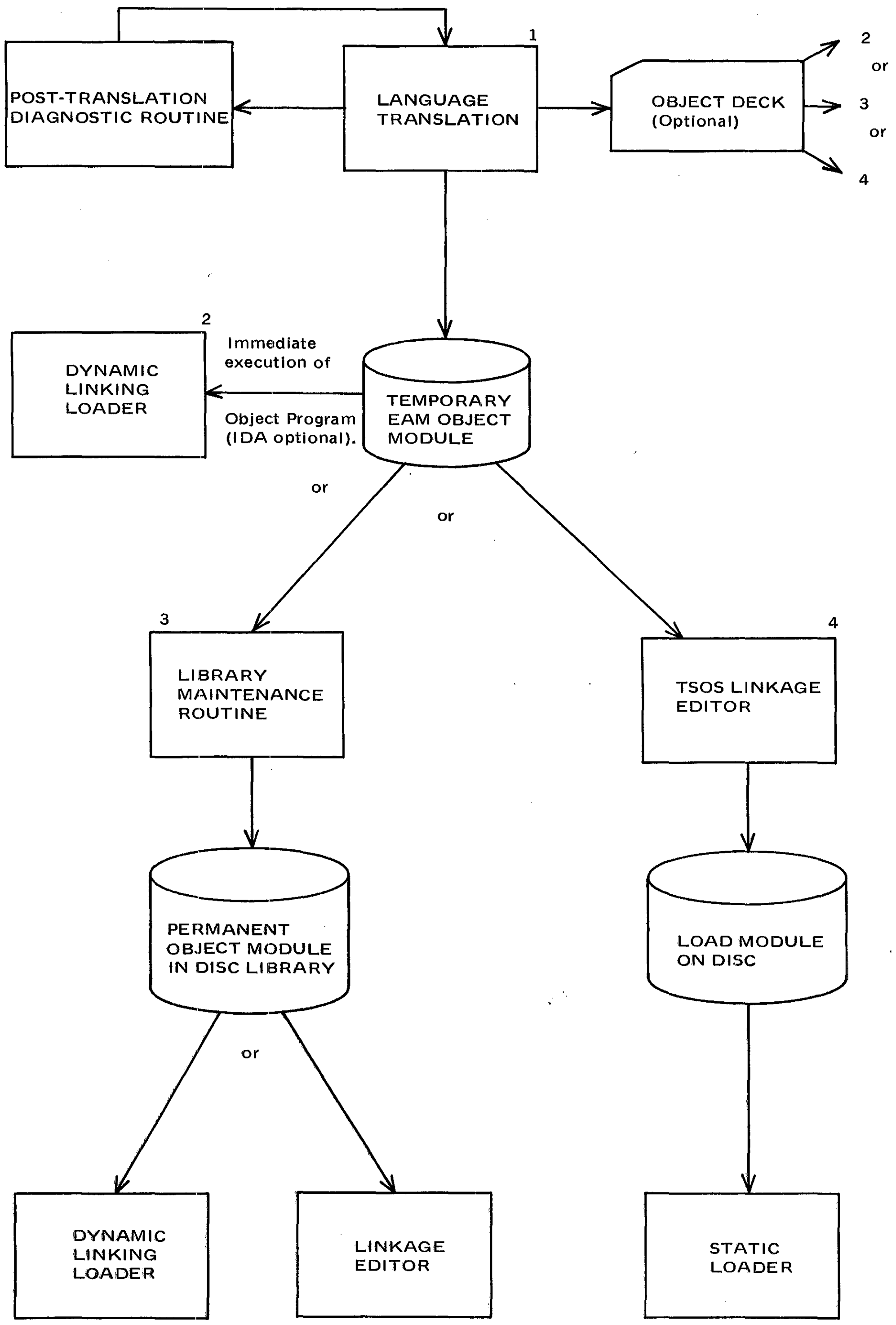
The conversational programmer may call on the post-assembly diagnostic routine, ADIAG, to check the diagnostics, if any, that the Assembler produced. For a description of the ADIAG program, refer to Chapter 6 of this manual.

The Assembler generates a permanent diagnostic file on disc if the user requests it at Assembly time with the command /PARAM ERRFIL=YES. This is an indexed sequential (ISAM) file for which the Assembler creates a catalog entry and allocates disc space; the filename is constructed by suffixing the name of the first CSECT (or the START statement) with the characters .ASSM.DIAG. For instance, if the first CSECT is named BAT, the Assembler will use the file name BAT.ASSM.DIAG for the diagnostic file.

The programmer can supply a filename for the diagnostic file. The paragraph entitled Naming the Diagnostic File in Chapter 6 fully describes the commands needed to do this.

INTERACTIVE DEBUGGING AID (IDA)

After achieving an error-free compilation, a programmer normally wants to check the program logic in one or more test runs, some of which will probably indicate the need for changes in the source program and consequent recompilation.



1
LANGUAGE TRANSLATION
POST-TRANSLATION DIAGNOSTIC ROUTINE
OBJECT DECK (Optional)
2
or
3
or
4
2
DYNAMIC LINKING LOADER
Immediate execution of Object Program (IDA optional).
TEMPORARY EAM OBJECT MODULE
or
or
3
LIBRARY MAINTENANCE ROUTINE
4
TSOS LINKAGE EDITOR
PERMANENT OBJECT MODULE IN DISC LIBRARY
LOAD MODULE ON DISC
or
DYNAMIC LINKING LOADER
LINKAGE EDITOR
STATIC LOADER

FIGURE 4-1. POST-COMPILATION ROUTINES, FLOWCHART.

The programmer may use the Interactive Debugging Aid (IDA) to monitor and debug the program in test execution runs.

A user who wishes to employ IDA for hands-on debugging of the object program must not only instruct the compiler to create an Internal Symbol Dictionary, but must also specifically instruct the Linkage Editor or the Dynamic Linking Loader to create a loadable program containing an Internal Symbol Dictionary (ISD) for use by IDA. If he does not do so, the ISD will not be completed and placed in the load module. To request its inclusion, the programmer should give the parameter IDA=YES in the /LOAD or /EXECUTE command to the Dynamic Linking Loader, or the parameter IDA=Y in the PROGRAM statement of the TSOS Linkage Editor.

The ISD records which the compilers create are preserved intact in the programmer's Object Module Library created by the Library Maintenance Routine, unless their exclusion is specifically requested.

When instructed, the Linkage Editor and Linking Loader will generate ISD symbol entries for every entry symbol appearing in the object module ESD records. The generation of this ISD data is distinct from the compiler's generation of ISD records. Those items in ESD records of COBOL object modules for which the Linkage Editor and Dynamic Linking Loader will generate ISD symbol entries are the program name and paragraph names (ESD types SD and LD, respectively). The items in ESD records of FORTRAN object modules for which the Linkage Editor and Dynamic Linking Loader generate ISD symbol entries are the program name, named common, and label definitions (ESD types SD, CM, and LD, respectively).

Note: A programmer may wish to compile a program with ISD records in the object module and test it, using IDA. After testing, the user may realize that the program contains no errors; therefore, there is no further use for the Symbol Dictionary. In this situation, the user need not recompile, simply link and load without the IDA parameter. The ISD records generated by the compiler will not be included in the loaded program.

IMMEDIATE EXECUTION OF AN OBJECT PROGRAM

The user can execute his object program directly after compilation by invoking the Dynamic Linking Loader. Because the compiler-generated EAM object module and the loadable program produced by the DLL are both on temporary disc files, immediate execution is best suited to the programmer who is repeatedly compiling, testing for logic errors, and recompiling.

The user can use one of the following command streams:

Command Stream A.

```
/EXEC BGCOB (or BGFOR)          compile a FILE command for
/FILE                           command for every
.                               file accessed by the object program
.
.
.
/EXEC *                         Load & execute object program (*
                                specifies object module in EAM
                                file)
```

Command Stream B.

/PARAM SYMDIC=YES	Instructs compiler to generate Symbol Dictionary
/EXEC BGCOB (or BGFOR) /FILE . . .	
/LOAD*,IDA=YES	Loads object module in EAM file and instructs DLL to complete Symbol Dictionary
/EXEC BGCOB (or BGFOR) /FILE . . .	
/LOAD*,IDA=YES	Loads object module in EAM file and instructs DLL to complete Symbol Dictionary
/. /. /.	Any IDA commands needed for debugging
RESUME	Initiates execution of object program

The programmer should note, however, that the Dynamic Linking Loader handles Class II programs only. Class I programs must be linked by the Linkage Editor and then loaded and executed.

Notes:

1. For a full description of the Dynamic Linking Loader, see the Utility Routines Reference Manual.

2. See Example C of the COBOL section of this manual for a session in which a user compiles his source program and executes it immediately.

DELAYED EXECUTION OF AN OBJECT PROGRAM

When the user is satisfied that the program contains no logic errors, a permanent, disc-resident loadable program can be created which can be executed without recompilation. There are two ways in which this can be done.

Library Maintenance Routine

The first is by using the Library Maintenance Routines to place the object module in a disc resident Object Module Library (OML). Object modules which are in an OML can be used as input to the Linkage Editor or to the Dynamic Linking Loader.

Before invoking the Library Maintenance Routine, the programmer must be sure to create a catalog entry and allocate disc space for the OML with the FILE command of DMS, because the LMR does not do this. Also, because LMR takes its commands from the SYSDTA file, the programmer must be sure that SYSDTA is directed to the source which will contain those commands. For instance, a terminal user who is compiling and then placing the object module in an OML needs to redirect SYSDTA away from the source-language file and back to the terminal so that LMR commands can be typed in should issue:

```
/SYSFILE SYSDTA=source-file
/EXEC BGCOB (or BGFOR)
/SYSFILE SYSDTA=(PRIMARY)   Redirect SYSDTA to terminal
/EXEC LMR
*CONTROL etc.  The system types * and the user types commands to LMR
```

Note: For a complete description of the Library Maintenance Routine, refer to TSOS Utility Routines manual.

Linkage Editor

Second, object modules produced by the compiler may also be used as input to the Linkage Editor, which constructs a loadable program from one or more object modules in its input stream and writes this program on a permanent disc file. The output of the Linkage Editor can be loaded and executed by the Static Loader. For a full description of the Linkage Editor, refer to the Utility Routines Reference Manual.

The Linkage Editor expects to find its control statements in the SYSDTA file; therefore, a programmer who is compiling and then linking must be careful after compilation to redirect SYSDTA away from the source-language file to the file or device which contains the Linkage Editor control statements. For example, a terminal user who wished to enter Linkage Editor control statements from the terminal should use the following command stream:

```
/SYSFILE SYSDTA=source-file-name
/EXEC BGCOB (or BGFOR)
/SYSFILE SYSDTA=(PRIMARY)
/EXEC TSOSLNK
Linkage Editor control statements
```

The user may use the FILE command of DMS to create a catalog entry and allocate space for the disc file which is to contain the loadable program. However, if the programmer does not do so the Linkage Editor will do it.

If the user decides to catalog and allocate a file, care should be taken to specify the filename correctly in the Linkage Editor PROGRAM statement.

If the file indicated in the PROGRAM statement of the Linkage Editor already contains text (e.g., from a previous execution of the Linkage Editor), it will be over written with the loadable program created in the current run.

> Note: See Example B at the end of the COBOL section of this manual for a session in which a user compiles a source program and then binds it with the Linkage Editor.

5. POST-COMPILATION DIAGNOSTIC ROUTINE (BDIAG)

INTRODUCTION

The Interactive Diagnostic Routine is a post-compilation program designed to provide the user at a remote terminal with diagnostic information concerning a Background COBOL or FORTRAN Compilation. It operates as an interactive, Class II program.

OPERATION AND USE

To use the Diagnostic Routine, the programmer proceeds as follows.

Before a compilation, the user must specify via the command:

```
/PARAM ERRFIL=YES
```

that the background compiler (COBOL or FORTRAN) is to generate a diagnostic file on disc.

The Diganostic disc file is accessed via the DMS Sequential Access Method (SAM). The first record of the file is a summary record, followed by a variable number of diagnostic detail records.

The conversational user at a terminal may at any time query the status of the diagnostic file for a particular compilation. The programmer can verify the existence of the diagnostic file by issuing either the command /FSTATUS with the appropriate filename or the BDIAG command STATUS.

As soon as the user has verified that the file exists, that is, that the file has been generated and can be accessed, the Interactive Diagnostic Routine and diagnostic commands can be invoked.

The Diagnostic Routine operates in a conversational mode as follows:

The command /EXEC BDIAG invokes the routine.

As soon as the routine takes control, the following messages will be typed out on the terminal:

```
BF BDIAG
BF DEFAULT IS FORTRAN
BF FOR COBOL TYPE C,'COMMAND
BF READY'*
```

The programmer may type in a command following *. For any command, at least one space must exist between operands. The maximum allowable length for a filename is 44 characters.

After processing a command, BDIAG types another *. At this point, the user may enter the next command.

When the programmer issues a STATUS command, the routine reads the summary record and types out the total count of messages, followed by two counts according to severity type. These totals summarize the relative success of the particular compilation. If there are any errors of severity type 2, it indicates that the Compile and Go mode has been inhibited. If there are any of type 1, the user has the prerogative of deciding whether or not to attempt executing the object program.

When the programmer issues a BDIAG PRINT command, the routine reads all the detail records in sequential order, but types only those error messages specifically requested. If the file contains no detail records (i.e., no error messages from the compiler), the routine will type out a reply stating NO ERRORS.

The programmer may interrupt and terminate execution of any Diagnostic command.

COMMAND NOTATION

In the command format, the notations braces { } and square brackets [ ] are used in accordance with the following conventions:

{ } Braces enclose two or more items from which one item must be chosen.

[ ] Square brackets enclose options which may be included or omitted, as required.

At least one space must appear between the command code and the operand, as well as between each word in the operand field. The last character of a command must be an end-of-text character.

The following symbols are used in this document to represent data in the system's response messages:

F	Special character that identifies a Post-Compilation Diagnostic Routine response message.
iiii	Error message index number.
SSSSS	COBOL source statement sequence number.
V	Severity code.

COMMAND DEFINITIONS

The four commands in this routine (STATUS, PRINT, HELP, and DEND) all have the following format:

Command Field	Operand Field
Command Name spelled out in capital letters or abbreviated to a single capital letter	One or more parameters

The operand field is optional for some commands.

STATUS Command

Provides a summary of the results of the compilation. It enables the user to decide whether to proceed to execute the object program, or request additional diagnostic information to determine the usefulness of the compilation.

```
User:        {STATUS}      diagnostic-filename
             {S     }
```

BDIAG Response:

```
F TOTAL OF ttttt DIAGNOSTIC MESSAGES
(F SEVERITY=0 FOR ttttt)
(F SEVERITY=1 FOR ttttt)
(F SEVERITY=2 FOR ttttt)
(F SEVERITY=3 FOR ttttt)
```

Where ttttt is the total number of messages in question. The last four lines will not be typed if the total number of diagnostic messages is zero.

Example:

```
User:        STATUS  TAXPROG.ERRFIL
```

BDIAG Response:

```
F TOTAL of 00015 DIAGNOSTIC MESSAGES
F SEVERITY=0 FOR 00011
F SEVERITY=1 FOR 00003
F SEVERITY=2 FOR 00000
F SEVERITY=3 FOR 00001
```

If there are no error messages, there will still be a response in the form:

```
F TOTAL OF 00000 DIAGNOSTIC MESSAGES
```

PRINT Command

There are four forms of the PRINT command. In the following format specifications iiiii represents index code, sssss represents source sequence number, and v represents severity type.

PRINT ALL

Lists all the diagnostic messages. (They also appear on the Diagnostic Listing generated by the COBOL Compiler.) Each message contains the index code, source sequence number, and severity type, plus a full explanation that may extend across several lines.

```
User:        {PRINT}      diagnostic-filename  ALL
             {P    }
```

BDIAG Response:

```
F      iiiii     sssss     v          explanation
                           (cont'd)
                           (cont'd)
F      iiiii     sssss     v          explanation
.      .         .                    .
.      .         .                    .
.      .         .                    .
.      .         .                    .
```

Example:

```
User:          PRINT          TAXPROG.ERRFIL   ALL

BDIAG Response:

F       21074     00030    2           INVALID RERUN
                                       OPTION
F       31128     00048                LEVEL NUMBER
                                       EXCEEDS TWO
                                       DIGITS
.       .         .                    .
.       .         .                    .
.       .         .                    .
.       .         .                    .
```

PRINT ERRORS

Lists all the diagnostic messages, typing out only the index code, sequence number, and severity type.

```
User          {PRINT}         diagnostic-filename   ERRORS
              {P    }         E

BDIAG Response:

F       iiiii     sssss    v
F       iiiii     sssss    v
.       .         .        .
.       .         .        .
.       .         .        .
.       .         .        .
```

Example:

```
User:          PRINT          TAXPROG.ERRFIL

BDIAG Response:

F       21074     00030    2
F       31128     00048    1
F       E2010              3
```

PRINT SEVERITY

Lists all the diagnostic messages associated with a specified line number, i.e., COBOL source statement number. Also, to list all diagnostic messages without any source statement numbers, i.e., null line numbers.

```
User:          PRINT          TAXPROG.ERRFIL LINE=00195

BDIAG Response:

F       71174     00195    1           PERIOD MISSING
                                       OR I WORD
                                       APPEARING ON T
```

Example:

```
User:        PRINT        TAXPROG.ERRFIL L=NULL

BDIAG Response:

F      21074        2        INVALID RERUN
                             OPTION
F      31128        1        LEVEL NUMBER
                             EXCEEDS DIGITS
```

For COBOL, the programmer must have a five digit line number. Leading zeros are allowed for COBOL line numbers.

If the programmer specifies the NULL option, BDIAG searches through the diagnostic file and prints all diagnostic messages associated with source statements which have no source sequence number.

HELP Command

Lists all the diagnostic commands.

```
HELP H
```

BDIAG Response:

```
F Description of each command.
```

DEND Command

Terminates the routine.

```
DEND
```

PROGRAM MESSAGES

If a command is syntactically wrong, the routine will respond with:

```
BF BAD COMMAND
BF RETYPE
BF READY*
```

If the programmer specifies a severity code outside the specified range, an error message in the following form will be typed out:

```
E ILLEGAL SEVERITY SPECIFIED
BF READY*
```

If the programmer specifies a wrong filename, the following error messages will occur:

```
F FILE HAS NOT BEEN ALLOCATED
E BRANCH TO OPEN EXIT 1
F PLEASE LOGOFF AND LOGON AGAIN
```

At this point the routine automatically terminates, and the user should LOGOFF and start again.

CONSIDERATIONS FOR USE

Programmers inquiring about a COBOL compilation need to type in C command before using the (PRINT filename line=sssss) command. Should the user decide to inquire about his FORTRAN compilation after inquiring about the COBOL compilation, the programmer should type the F command. Alternatively, the diagnostic routine may be terminated and restarted. See the example following.

If the first compilation about which the programmer wishes to inquire is a FORTRAN compilation, the F command need not be typed.

In addition, the user can use the STATUS command of BDIAG to determine whether the diagnostic file has been built. If the file has been built the programmer will receive normal response to STATUS; otherwise, the same error message will be received as if the programmer specified an incorrect filename (see item 3 in the foregoing explanation of Program Messages).

```
/PARAM LIST=YES,ERRFIL=YES
/SYSFILE SYSDTA=EK.COBSRC (EK.COBSRC IS A FILE CONTAINING THE
/EXEC BGCOB      CONTAINING THE SOURCE PROGRAM NAMED "POLITICS".
%L001 PROGRAM LOADING     "POLITICS".
 32A0 COMPILATION INITIATED (BGCOB VERSION=035B)
 32AA COMPILATION COMPLETED WITH SERIOUS ERRORS
%EB001 SPOOLOUT INITIATED FOR TSN=4862 ID=HPBLS857
%      PRINT FILE=00034
/SYSFILE SYSDTA=EK.FORT
/EXEC BGFOR
%L001 PROGRAM LOADING
 A      TSOS FORTRAN BACKGROUND COMPILER **VER A10**
%EB001 SPOOLOUT INITIATED FOR TSN=4865 ID=HPBLS857
%      PRINT FILE=00027
/SYSFILE SYSDTA=(PRIMARY)
/EXEC BDIAG
%L001 PROGRAM LOADING
 BF BDIAG
 BF DEFAULT IS FORTRAN
 BF FOR COBOL TYPE C.  COMMAND
 BF READY
*STATUS POLITICS.ERRFIL
 F TOTAL OF 00001 DIAGNOSTIC MESSAGES
 F SEVERITY=2 FOR 00001
 BF READY
*C.
 BF READY
*PRINT POLITICS.ERRFIL ERRORS
 F B1006 00048       2
 BF READY
*STATUS EKFORT.ERRFIL
 F  NO DIAGNOSTIC ERRORS
 BF  READY
*F.
 BF  READY
*PRINT EKFORT.ERRFIL ERRORRS
 F  NO DIAGNOSTIC ERRORS
 BF READY
*DEND
/LOGOFF BUT
%C E420 LOGOFF AT 1130 ON 01/21/71, FOR TSN 4901.
%C E421 CPU TIME USED: 0001.5939 SECONDS.
```

6. POST-ASSEMBLY DIAGNOSTIC ROUTINE (ADIAG)

INTRODUCTION

The Post Assembly Diagnostic Routine is a post-assembly program designed to provide the programmer at a remote terminal with diagnostic information about a particular execution of the Assembler. It operates as a Class II program which can be executed conversationally and nonconversationally.

OPERATION AND USE

To use the Post-Assembly Diagnostic Routine, the programmer procedes as follows:

Before an assembly, the user must specify in the command:

```
/PARAM ERRFIL=YES
```

that the Assembler is to generate a disc resident diagnostic file.

The diagnostic file is accessed by the DMS Indexed Sequential Access Method (ISAM).

The conversational programmer at a terminal may at any time query the status of the diagnostic file for a particular assembly. The user can verify the existence of the diagnostic file by issuing either the FSTATUS command with the appropriate file name or the ADIAG STATUS command described below.

As soon as the programmer has verified that the file exists and can be accessed, the Interactive Diagnostic Routine, ADIAG, may be invoked and diagnostic commands may be entered.

During the conversational session, the user has the option of using the Diagnostic Program in two different modes: the terminal mode and the printer mode.

Terminal Mode

This is the standard conversational mode. The commands are accepted from SYSDTA and the responses are written to SYSOUT. Both these files are directed to the terminal.

Printer Mode

In this mode, the commands are still accepted from SYSDTA. However, the results are put out to a high-speed printer at task termination. This mode permits the user to extract or construct selected data to produce, in effect, hand-tailored assembly listing data.

At the beginning of a session, ADIAG is in the terminal mode. The programmer can switch to the printer mode by using the MODE command.

When ADIAG is in the printer mode, the commands are scanned as they are entered. They are not immediately executed but are placed in an Active List which is a list of commands to put out data on the printer after the user has ended the session with the Assembler Diagnostic Routine.

Thus, as long as the Diagnostic Program is in the printer mode, it is constructing a list of commands for later execution. None of the commands in the list will be executed until after the current session with the Diagnostic Program is ended. At any time during the session, the list of commands that are in the list may be displayed at the terminal (VERIFY command). Also, any or all of the commands may be deleted from the list of commands to be executed (REMOVE command).

Invoking ADIAG

The command /EXEC ADIAG invokes the routine. As soon as ADIAG has been loaded, it displays an asterisk at the terminal to indicate that it is ready for the programmer to enter a STATUS command. The STATUS command enables the programmer to specify the particular diagnostic file to be interrogated.

After processing a command, ADIAG displays another *. At this point, the user may enter his next command.

Command Interruption

The Diagnostic Program is constructed so that the programmer can interrupt and terminate execution of any diagnostic command by pressing the BREAK key. This interrupts the Diagnostic Program and puts the user in communication with the Executive. The programmer can return to the Diagnostic Program by typing the Executive command RESUME, in which case execution continues as if the program had not been interrupted; or the user can type the Executive command INTR, in which case the Diagnostic Program will abort any command in progress and expect the programmer to enter the next command.

The Diagnostic Program can be used conversationally from a terminal so that the programmer can control processing, one command at a time. It can also be used nonconversationally from a sequential file or system card reader. The Diagnostic Program gets its commands from a logical sequential file which is equated with SYSDTA. Output in the form of responses and messages is directed either to the user's terminal in the case of a conversational task, or the printer in a nonconversational task.

STRUCTURE AND USE OF THE DIAGNOSTIC FILE

Structure

The Diagnostic File is created at assembly time; for a description of how to request its creation, refer to the section in Chapter 3 entitled Use of TSOS PARAM Command for Assembly. The file contains all the data that appears in the normal Assembly listing, plus some additional summary information for use by the Diagnostic Program. The file is constructed as an indexed sequential file for more efficient accessing by ADIAG.

Naming the Diagnostic File

The Assembler provides a default filename for the diagnostic file, unless the programmer wishes to supply a name for the diagnostic file.

The default file name is constructed by prefixing the name of the first CSECT in the assembly to the characters .ASSM.DIAG (i.e., csectname.ASSM.DIAG). After the user has invoked ADIAG, a STATUS command must be issued specifying the csect-name. ADIAG then finds the correct file for the programmer. (In the case where the first CSECT is unnamed, the Assembler names the diagnostic file ASSM.DIAG and the STATUS command needs no operand.)

If the user wishes to supply a filename for the diagnostic file, a FILE command must be issued in the following format before the programmer invokes the Assembler:

```
/FILE filename,LINK=DIAGFILE,SPACE=(primary,secondary)
```

The SPACE parameter is not required if the file has already been allocated space. After the assembly is completed and before invoking ADIAG, the programmer must issue another FILE command with the following format:

```
/FILE filename,LINK=DIAGLINK
```

The programmer then invokes ADIAG and issues a STATUS command, either specifying as an operand the name of the first CSECT, or, if the CSECT was unnamed, leaving the operand field blank.

DIAGNOSTIC PROGRAM COMMANDS

ADIAG commands provide the means by which a programmer specifies the diagnostic inquiries to be made. Commands are normally entered from a terminal keyboard.

Commands usually consist of four fields:

1. Label Field

The label field is an optional field. It provides a means for the programmer to reference a particular command that has been entered earlier in the session while in the printer mode. The label field consists of a symbol preceded by a period. A symbol consists of from 1 to 7 characters, the first of which must be a letter and the remaining are either letters or decimal digits.

2. Command Operation Field

The command operation field identifies the function to be performed. This field must always be present.

3. Command Operand Field

The command operand field identifies the particular types of data about which the inquiry is being made. For most commands, the operand field is optional and, when not present, a pre-specified default value will be assumed.

4. ETX Character

The ETX character must be used to terminate the entering of a command when the commands are being entered from a terminal.

Conventions for Command Description

Braces, { } enclose two or more parameters from which a choice must be made. Brackets, [ ] enclose options which may or may not be included. At least one space must appear between the command code and the operand, as well as between each word in the operand field. The last character of a command must be the ETX character.

.label, represents a command label field Δ represents a space.

Command Definitions

STATUS Command

Determines whether the diagnostic file exists and, if it does, summarizes briefly the results of the Assembly. It must be issued before all other commands except HELP, DEFINE and END. It enables the programmer to, decide whether to issue further diagnostic commands or to go ahead and execute the object module.

$$\text{[.label]} \qquad \left\{\begin{matrix}\text{STATUS}\\ \text{S}\end{matrix}\right\} \qquad \left[\left\{\begin{matrix}\text{csect-name}\\ \text{csect-name\#password}\end{matrix}\right\}\right]$$

Csect-name

Is the name of the first CSECT in the assembled module.

password

The password, if one is required, for access to the file.

COMMAND EXECUTION

ADIAG tries to open the file named. If it cannot do so, then a message is printed on SYSOUT. If the file is opened successfully, ADIAG displays a summary giving (1) the name of the file opened, (2) date and time the file was created, (3) whether the file has been accessed by the Diagnostic Program before (indicated by the word OLD or NEW), (4) an error severity code, and (5) a count of the number of statements in error.

User:	STATUS PROG1
System Response:	NEW FILE-PROG1.ASSM.DIAG CREATED 06/04/70 08:56:49 SEVERITY=1 NO. OF FLAGGED STATEMENTS=000012 NUMBER OF ERROR FLAGS: 14

When the programmer wishes to have diagnostics written to a specific file, rather than to the standard diagnostic file which the Assembler names csect.ASSM.DIAG, two FILE commands must be issued. The following example illustrates how to accomplish this:

```
①/SYSFILE SYSDTA=EK.1
②/FILE EKERR,LINK=DIAGFILE,SPACE=(12,3)
③/PARAM ERRFIL=YES
 /EXEC ASSEMB
 %L001 PROGRAM LOADING
  VERS. 0009 OF TSOS ASSEMBLER READY
  FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
④/SYSFILE SYSDTA=(PRIMARY)
⑤/FILE EKERR,LINK=DIAGLINK
⑥/EXEC ADIAG
 %L001 PROGRAM LOADING
⑦*STATUS TEST
  NEW FILE--EKERR
  CREATED 11/24/70 13:52:52 SEVERITY=0 # OF FLAGGED STATEMENTS = 000000
  NO ERROR FLAGS
 *END
```

①	Direct SYSDTA to Assembly language source program.
②	FILE command for programmer named diagnostic file. Note linkname is DIAGFILE.
③	Instruct Assembler to write the diagnostic file
④	Re-direct SYSDTA to terminal in order to enter ADIAG commands.
⑤	Issue second FILE command for diagnostic file. This time, linkname must be given as DIAGLINK.
⑥	Invoke ADIAG.
⑦	Issue STATUS command specifying name of first CSECT in assembled module.

FLAGS Command

Gives a count by flag type of the Assembly errors. It gives the programmer a more detailed picture of the results of the Assembly.

$$[.\text{label}] \qquad \begin{Bmatrix} \text{FLAGS} \\ \text{F} \end{Bmatrix}$$

COMMAND EXECTUION

The diagnostic file is interrogated and each general flag is listed with a count of the number of times this flag appears.

EXAMPLE

```
User:           FLAGS

System
Response:       A=002, M=017, U=003
```

LINES Command

Lists the statement numbers and associated error flags of all statements which have errors.

$$[.\text{label}] \qquad \begin{Bmatrix} \text{LINES} \\ \text{L} \end{Bmatrix}$$

COMMAND EXECUTION

The diagnostic file is interrogated and each line that has an error flag is listed by statement number. The flag references for the line are listed to the right of the statement number.

EXAMPLE

```
User:           L

System
Response:       00334 U       00428AA       00437U
```

NUMBERS Command

Lists statement numbers by flag type for all statements having the flag types which are specified in the command operand field. If no operand is specified, the statement number of all statements containing error flags will be listed.

$$[.\text{label}] \qquad \begin{Bmatrix} \text{NUMBERS} \\ \text{N} \end{Bmatrix} \quad [\text{f1 f2 f3 f4}]$$

The f's may be either the general or detailed flags used by the Assembler to flag statements in error, and are specified by the user when he is interested in a particular type or subset of errors. The f's may appear in any order and may be redundantly entered. The operand field may have up to 20 entries.

COMMAND EXECUTION

The diagnostic file is interrogated and each specified flag listed with the line numbers where it occurs.

EXAMPLE

```
User:         NUMBERS     A U T

System
Response:     A 00428
              T NONE
              U 00334     00437
```

TAGS Command

Lists all undefined and multiply-defined symbols and, optionally, their associated cross-reference data. If no operand field is present, both multiply-defined and undefined symbols are listed; however, no cross-reference data is given.

$$[.\text{label}] \quad \left\{\begin{array}{l}\text{TAGS}\\ \text{T}\end{array}\right\} \quad [\text{M}]\ [\text{U}] \quad \begin{array}{l}\text{X}\\ \text{XREF}\end{array}$$

M

Requests that all multiply-defined symbols be listed.

U

Requests that all undefined symbols be listed.

X or
XREF

Lists cross-reference data for any M or U symbols listed.

The operands can appear in any order.

COMMAND EXECUTION

The diagnostic file is interrogated and the requested data is listed. If no cross reference data was requested, then all multiply-defined symbols are listed first and then all undefined symbols. If cross-reference data was requested, then each symbol is followed by a list of statement numbers which reference the symbol. Where the referencing statement is the defining statement, the statement number is followed by an asterisk (*).

EXAMPLES

For the following examples assume:

1. Symbol FIELDA is multiply defined by statements 00605 and 00900 and referenced by statement 00910.

2. Symbol AREA2 is multiply defined by statements 01000 and 01500 and referenced by statement 00905.

3. AREA1 and FIELDB were undefined symbols referenced by statements 01400 and 01410, respectively.

```
User:          T

System
Response:      M AREA1,'FIELDA
               U AREA1,'FIELDB

User:          TAGS X

System
Response:      M AREA 2 01000*,'00905,'01500
               M FIELDA 00605*,'00900,'00910
               U AREA1 1400
               U FIELDS 01410

User:          TAGS U X

System
Response:      AREA1 01400
               FIELDB 01410
```

XREF (Cross-reference) Commands

Lists the cross-reference statement numbers for the symbols specified in the operand field of the command.

$$[.\text{label}] \qquad \begin{Bmatrix}\text{XREF}\\ \text{X}\end{Bmatrix} \qquad \text{S1 S2 S3} \ldots$$

S1, S2, and S3 are symbols for which the programmer requires cross-reference data. Each symbol in the operand list is separated by at least one space. At least one symbol must appear in the list.

COMMAND EXECUTION

The diagnostic file is interrogated and each symbol designated is listed along with its associated cross-reference listing data.

EXAMPLE

```
User:          XREF          AREA1

System
Response:      AREA1         00100*00212 00257
```

PRINT Command

Lists a specified statement or a specified range of statements as they would appear on the assembly listing.

$$[.\text{label}] \qquad \begin{Bmatrix}\text{PRINT}\\ \text{P}\end{Bmatrix} \qquad \begin{Bmatrix}\text{s1}\\ \text{l1}\end{Bmatrix} \qquad \begin{Bmatrix}\text{s2}\\ \text{l2}\\ +\text{N}\end{Bmatrix}$$

s1 or l2

Specifies the statement at which to begin printing. Printing will begin at the line whose tag is s1 or whose statement number is l1.

s2 or l2

Specifies the statement with which to end printing. Printing will end with the printing of the line:

1. whose tag is s2, or
2. whose statement number is l2, or
3. is the Nth line beyond the initial line.

Note: If the s2 or l2 parameter is omitted, then the line indicated by s1 or l1 is the only one printed.

COMMAND EXECUTION

The statements specified are read from the diagnostic file, formatted and displayed. In general, each Assembler statement will require two lines for printing, with line 1 containing the first 65 characters and line 2 the remainig characters of the statement. The exact format of the printouts will vary depending on the permitted line sizes for the particular remote terminal equipment being used.

EXAMPLE

```
User:          P 03125

System
Response:      081DC D2 07 A2C7 A230 093A7 09310 03123
               DNRERPL MVC AREPLACE(8),A8ZER
```

DEFINE Command

Lists the definition of the specified error flags.

$$[.\text{label}] \quad \begin{Bmatrix}\text{DEFINE}\\ \text{D}\end{Bmatrix} \quad \text{f1 f2}\ldots$$

f1. . . fn are the flag characters used by the Assembler on a listing to indicate source statement errors. Both the single letter flags and the detailed flag indicators may be specified in the command. The maximum number of flag entries permitted in a single command is 20.

COMMAND EXECUTION

The definition of each of the flags listed in the command is printed on the terminal. The definitions will be printed in the same order as they are requested in the command operand field.

EXAMPLE

```
User:          DEFINE U

System
Response:      U1             UNDEFINED SYMBOL.
```

HELP Command

This command provides a conversational programmer's manual for the user. The command may be invoked to get: (1) a list of the Diagnostic Program commands or (2) a description of one or more commands. The user may enter only the operation code and the system will then write a list of the Diagnostic Program commands. If one or more command names are entered in the operand field, then a description is written for each command named.

$$[.\text{label}] \qquad \begin{Bmatrix}\text{HELP}\\ \text{H}\end{Bmatrix} \qquad [\text{name 1 name2}\ldots]$$

If the operand field is omitted, a list of the Diagnostic Program commands is printed without any additional description. If the names (operation codes) of one or more commands are given (each name must be followed by at least one space), then a description is written for each command named.

COMMAND EXECUTION

The command is scanned and the appropriate messages written out.

EXAMPLES

User: HELP LINES

System
RESPONSE: (.label) LINES

MODE Command

Places the Diagnostic Program in either the terminal mode, the printer mode, or both mode. It determines the mode in which the Diagnostic Program will operate until either an END or subsequent MODE command is processed.

$$[.\text{label}] \qquad \begin{Bmatrix}\text{MODE}\\ \text{M}\end{Bmatrix} \qquad \left[\begin{Bmatrix}\text{TERMINAL}\\ \text{T}\\ \text{PRINTER}\\ \text{P}\\ \text{BOTH}\\ \text{B}\end{Bmatrix}\right]$$

If no parameter field is present, then TERMINAL mode is set.

COMMAND EXECUTION

The Diagnostic Program is set to the requested mode. The command is ignored if the program is already in the mode indicated in the operand field.

EXAMPLE

User: MODE T

System
Response: (sets execution mode to TERMINAL mode)

VERIFY Command

Lists at the terminal those commands that have been entered into the Active List as a result of the Diagnostic Program having been placed in the printer mode. The command will list either all commands in the Active List or all commands specifically named (by their labels) in the operand field.

```
              {VERIFY}
[.label]      {V     }      [.label1 .label2 . . .]
```

If the operand field is missing, all commands in the Active List will be listed at the terminal. Otherwise, the commands identified by their label field will be listed at the terminal. A maximum of 5 label entries is permitted in the operand field.

COMMAND EXECUTION

The Active List is accessed and then the commands are listed.

EXAMPLE

User:	VERIFY .COM1
System Response:	.COM1 LINES

REMOVE Command

Deletes commands from the Active List. It removes all commands or specifically named commands from the list.

```
              {REMOVE}
[.label]      {R     }      [.label1 label2 . . .]
```

If the operand field is not present, all commands in the Active List are deleted. If the operand field contains one or more command labels, then the commands in the Active List identified by those labels are deleted. A maximum of 5 label entries is permitted in the operand field.

COMMAND EXECUTION

The Active List is scanned and the appropriate commands are deleted from the list.

EXAMPLE

User:	REMOVE .COM,BLANK
System Response:	(removes commands labeled .COM and .BLANK from the active list.)

END Command

Terminates the execution of the Diagnostic Program and can also be used to request that a standard Assembly listing be constructed and printed from the diagnostic file.

$$[.\text{label}] \qquad \begin{Bmatrix} \text{END} \\ \text{E} \end{Bmatrix} \qquad \left[\begin{Bmatrix} \text{LISTING} \\ \text{L} \end{Bmatrix}\right]$$

If no operand field is present, then the Diagnostic Program does not produce an Assembly listing. If present, the standard Assembly listing printed is printed in a background mode.

COMMAND EXECUTION

When this command is entered, the Diagnostic Program creates and enters a background task if either a listing was requested or there are commands in the Active List. After establishing the background batch task (if necessary), the Diagnostic Program terminates.

INDEX

Title	TSOS Language Processor Programming Reference Manual
Document No.	DJ-008-2-00
Date	July 1971

RCA Computer Systems

Your comments and suggestions will help us to furnish publications that are more useful to you.

Is this publication:

Complete in its coverage?

Logically organized?

Technically accurate?

Easy to understand?

Other comments (Use additional page if necessary).

Name ____________________ Street or Box No. ____________________

Job Title ____________________ City ____________________

Company ____________________ State __________ Zip ____________________

Cut Along Line

Fold

FIRST CLASS
PERMIT NO. 16
CAMDEN, N. J.

BUSINESS REPLY MAIL – no postage necessary if mailed in the United States

Postage will be paid by addressee

RCA | COMPUTER SYSTEMS
DATA PROCESSING DIVISION
CAMDEN, N. J. 08101

ATTN: Marketing Publications
Bldg. 204-2, Cherry Hill

Fold

Publications Purchase Order

RCA|Computer Systems Division
Camden, N. J. 08101

RCA

Order Number

Item No.	Quantity Ordered	Ordering Number	Description or Title of Material	Complete if Publications Are To Be Purchased	
				Unit Price	Totals
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

Ship To:

Bill To: (Complete if Publications Are To Be Purchased)

Customer P.O. #

Sub Total	
Indicate Applicable Sales Tax	
Total Cost	

Check Appropriate Block:

Bill My Company [ ]

Remittance Enclosed [ ]

If Publications are to be purchased forward this Form and any enclosures to:

RCA|Computer Systems Division
Reproduction Services
Camden, N. J. 08101

Publications purchased will be furnished subject to all terms and conditions stated on the reverse side of this Form.

Ship via

Date Required

Authorized Signature

Date

All other requests: Forward this form to the nearest RCA District Office.

27-41-002

9/70

Terms And Conditions
Applicable To The Sale of RCA Publications and Forms

RCA

PRICES

All prices are subject to change or withdrawal without notice and all shipments will be billed at prices in effect on date of shipment. Unless otherwise specified or required by law, all prices will be billed exclusive of state and local sales and similar taxes, and such taxes will appear as additional items on invoices.

TRANSPORTATION

All shipments will be f.o.b. destination.

On shipments where Purchaser requests transportation involving expenses beyond those involved on transportation normally selected by RCA, the Purchaser will be responsible for payment of such extra costs.

RCA reserves the right to ship from any location subject to the foregoing transportation terms.

DELIVERIES

It is the desire of RCA to meet requested delivery schedules. However, RCA shall not incur any liability due to any delay or failure to deliver for any reason. Any delivery indication furnished by RCA only represents the best estimate of the time required to make shipment. The delivery of part of any order shall not obligate RCA to make further deliveries, and RCA reserves the right to decline servicing any order in whole or in part.

Of necessity, inventories and current production must be allocated in such a manner as to comply with applicable Government regulations. In the absence of such regulations, RCA reserves the right to allocate inventories and current production when, in its opinion, such allocation is necessary.

TERMS OF PAYMENT

Invoices shall be rendered at time of shipment and shall be payable net 30 days from date of shipment.

Partial shipments will be invoiced as made, and payments therefor are subject to the above terms.

GENERAL

In no event shall RCA be liable for indirect, consequential or special damages.

Information furnished by RCA is believed to be accurate and reliable. However, no responsibility is assumed by RCA for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of RCA.

This Agreement shall be governed by the laws of the State of New York and constitutes the entire Agreement between the parties with respect to the subject matter hereof. It shall prevail regardless of any variation in the terms and conditions of any other submitted by the Purchaser.