GC28-6470-2
File No. S370-20

Program Product

IBM OS/VS COBOL Compiler and Library General Information

Program Numbers 5740-CB1 (Compiler and Library)
5740-LM1 (Library Only)

Release 2.4

IBM

GC28-6470-2
File No. S370-20

Program Product

IBM OS/VS COBOL Compiler and Library General Information

Program Numbers 5740-CB1 (Compiler and Library)
5740-LM1 (Library Only)

Release 2.4

Third Edition (August 1983)

This is a major revision of, and makes obsolete, GC28-6470-1.

This edition applies to Release 2.4 of OS/VS COBOL Compiler and Library, Program Product 5740-CB1 (Compiler and Library), and 5740-LM1 (Library only), and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface. Because the technical changes in this edition are extensive and difficult to localize, they are not marked by vertical bars in the left margin.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest <u>IBM System/370 and 4300 Processors Bibliography</u>, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

This publication contains information to aid data systems planners and analysts in evaluating the IBM program product, OS/VS COBOL Compiler and Library.

COBOL (COmmon Business Oriented Language) is an English-like programming language used for commercial data processing. It is developed by the Conference On DAta SYstems Language (CODASYL). In the USA, the standard of the language is American National Standard COBOL, X3.23-1974, as approved by the American National Standards Institute (ANSI).

INDUSTRY STANDARDS

The OS/VS COBOL Release 2.4 Compiler and Library is designed according to the specifications of the following industry standards, as understood and interpreted by IBM as of April 1976:

- The highest level of American National Standard COBOL, X3.23-1974 (excepting the Report Writer module). American National Standard COBOL, X3.23-1974 is compatible with and identical to International Organization for Standardization/Draft International Standard (ISO/DIS) 1989-COBOL.
- The highest level of American National Standard COBOL, X3.23-1968. American National Standard COBOL, X3.23-1968, is compatible with and identical to ISO/R 1989-1972 Programming Language COBOL.

A number of IBM extensions are also implemented.

BOOK ORGANIZATION

Included in this manual are brief descriptions of the OS/VS COBOL Compiler and Library features, information on compatibility, and system requirements. The manual is intended as an aid to evaluation and planning for use of the product; it is not intended to be used as a specifications manual. Specifications for the OS/VS COBOL program product are given in Program Product Specifications: OS/VS COBOL Compiler and Library, GC28-6472.

RELATED MATERIAL

IBM OS/VS COBOL Compiler and Library Programmer's Guide, SC28-6483

IBM VS COBOL for OS/VS, GC26-3857

IBM OS COBOL Interactive Debug and (TSO) COBOL Prompter, General Information, GC28-6454

VS COBOL II

For information about VS COBOL II, see VS COBOL II General Information, GC26-4042.

SUMMARY OF AMENDMENTS

RELEASE 2.4, AUGUST 1983

NEW PROGRAMMING FEATURE

Information about the MIGR compiler option has been added. MIGR flags major COBOL language elements that are no longer supported or are supported differently by the VS COBOL II Compiler, Program Number 5668-958.

RELEASE 2, SEPTEMBER 1976

NEW PROGRAMMING FEATURE

The text has been updated to reflect new OS/VS COBOL features, which include:

- Expanded Language Capabilities in support of the 1974 American National Standard COBOL
- Enhanced VSAM Support
- Expanded Physical Sequential Files through QSAM
- Added Communication Support
- Expanded Library Facilities
- User-Defined Collating Sequences for entire programs
- Federal Information Processing Standard for either the 1968 or 1974 American National Standard COBOL

CONTENTS

FIGURES

OS/VS COBOL OVERVIEW

This manual describes the OS/VS COBOL Release 2.4 Compiler and Library, an IBM program product that offers expanded language capabilities through support of 1974 Standard COBOL. OS/VS COBOL also provides advanced programming facilities meant to reduce program development time and to aid in increasing programmer productivity. OS/VS COBOL Compiler features are:

Expanded Language Capabilities: The OS/VS COBOL Release 2 compiler accepts source language in support of the 1974 Standard. It also accepts source language in support of the 1968 Standard. IBM extensions are also supported. The additional language capabilities make possible programming applications not feasible previously.

Virtual System Support, through OS/VS1 and OS/VS2, and the CMS component of VM/370, makes use of the performance improvements and large storage capacity of the Virtual Systems.

Advanced Program Applications are possible with these features:

- Enhanced VSAM Support—VSAM is a high-performance VS access method with a high degree of data security. The following file organizations are supported:
  - VSAM Sequential Files—using entry sequenced data sets.
  - VSAM Indexed Files—using key sequenced data sets, and, additionally, for Release 2, alternate indexes for record retrieval.
  - VSAM Relative Record Files—using relative record data sets.
- Expanded Physical Sequential File Capabilities—through QSAM, the following processing capabilities are available:
  - Files can be extended.
  - Input and/or output requests can be tested for success or failure.
  - The record area is available when the file is opened.
  - Logical page formatting of the printed logical output page can be specified through the source program.
- Added Communication Support—with the following features available through OS/VS COBOL:
  - Automatic scheduling of the COBOL Communication program upon demand.
  - Queues made available/unavailable under COBOL program control.
  - Multiple destinations can be specified.
- Expanded Library Facilities—with the following features:
  - References to library members are allowed at any point in the COBOL program.
  - Multiple libraries can be specified.
  - Replacement library text can be a literal, a word, or any COBOL character string(s).

- Powerful Data Manipulation—through the following COBOL statements:
  - INSPECT Statement—counts and/or replaces one or more characters in a data item; allows full or partial initialization of such items.
  - STRING/UNSTRING Statements:—multiple subfields can be combined into a single field; a single field can be separated into multiple subfields.
- Extended Computational Facilities—allows multiple receiving fields in any arithmetic statement.
- User-Defined Collating Sequences—can be specified for an entire program, for physical sequential data files, or for Sort/Merge comparisons.
- Merge Facility—The MERGE verb enables two or more identically sequenced input files to be combined into a single output file by specifying a set of keys. Both standard sequential and sequentially accessed VSAM files can be designated as input or output.
- System/370 Device Support—including optical character readers and large-capacity high-speed disk facilities. High-performance System/370 devices also speed up execution and allow advanced applications.
- Dynamic Subprogram Linkage—at execution time, user subprograms can be fetched and loaded dynamically. The storage assigned to such a subprogram can also be released when the subprogram is no longer needed.

Program Development Aids are provided through COBOL language that eases former programming rules, through high-level debugging features, and through standardization aids.

- Eased Programming Rules—give added COBOL capabilities and make program development simpler:
  - Table handling rules allow mixed indexes and literal subscripts, as well as negative literals in the SET statement.
  - Level 77 items in Data Division entries can follow level 01 items.
  - Relaxed punctuation rules.
  - Page ejecting comment lines simplify page-by-page format control of the source program listing.
- COBOL Source Program Debug Language—allows selective monitoring of file-names, cd-names, procedure-names, and identifiers. Both a compile-time switch and an object-time switch are available to activate or deactivate the debugging language.
- Federal Information Processing Standard (FIPS) Flagger—ensures that OS/VS COBOL programs can be written to conform to a selected level of either the 1972 or the 1975 Federal Information Processing Standard.
- Migration Flagging—allows the COBOL user to identify significant OS/VS COBOL language that is either no longer supported or is supported differently by the VS COBOL II compiler.

- Interactive Capabilities—under OS/VS2 TSO or VM/370 CMS, compiler output can be directed to a terminal, and COBOL background debugging facilities are available at a terminal. (In addition, the TSO COBOL Prompter Program Product is available to the TSO COBOL user, and the OS COBOL Interactive Debug Program Product is available to the TSO or CMS COBOL user.)

- WHEN-COMPILED Special Register—provides a means of associating a compilation listing with both the object program and the output produced at execution time.

- Lister Facility—provides specially formatted listing with embedded cross-references for increased intelligibility and ease of use. Reformatted source deck is available as an option.

- Verb Profiles—facilitates identifying and locating verbs in the COBOL source program. Options provide verb summary or verb cross-reference listing which includes verb summary.

- Execution-Time Statistics—maintains a count of the number of times each verb in the COBOL source program is executed during an individual program execution.

- Background Symbolic Debug—during program execution, COBOL-formatted snapshots and maps of the Data Division can be obtained, either at specified points during execution or at abnormal termination. Any number of debugging runs can be executed without recompilation. No COBOL source language changes are needed.

- Flow Trace—shows program flow up to the point of abnormal termination. The path of execution within and between user-specified modules can be traced. No COBOL source changes are needed.

- Syntax-Checking Compilation—reduces compile time significantly by scanning the source code for syntax errors but (conditionally or unconditionally) produces no other compiler output.

- Statement Number Option—provides information about the COBOL statement being executed at abnormal termination.

Efficient Object-Time Performance can be achieved with OS/VS COBOL through the following features.

- Optimized Object Code—can reduce generated Procedure Division code. Programs are divided into 4K-byte procedure blocks. Register assignment is optimized.

- COBOL Library Management Facility—allows COBOL object programs running in separate regions or partitions to save virtual storage by sharing COBOL library subroutine modules.

- Dynamic Subprogram Linkage—gives object-time control of virtual storage resources. When a subprogram is no longer needed, the storage it occupies can be freed.

- System/370 Instruction Generation—System/370 instructions save object program space and speed up execution.

- Standard Block Specification—is allowed at object time. Use of the Fixed Standard Block option (particularly for direct access storage devices having the Rotational Positional Sensing feature) results in improved input/output performance.

Productive Compile-Time Performance is easy to achieve with OS/VS COBOL. The following feature optimizes performance:

- Speedy Sorted Cross-Reference—alphabetized cross-reference listings make it easier to find user-specified names.

OS/VS COBOL COMPILER FEATURES

The OS/VS COBOL Compiler is a program product that offers the following capabilities: expanded language capabilities, advanced program applications, program development aids, efficient object-time performance, and productive compile-time performance, all of which are described on the following pages.

EXPANDED LANGUAGE CAPABILITIES

The OS/VS COBOL Compiler accepts source language in support of the 1974 COBOL Standard. Through this language support, the programmer can make use of the following programming advancements:

- Enhanced VSAM Support
- Expanded Physical Sequential File Capabilities
- Added Communication Support
- Expanded Library Facility
- Powerful Data Manipulation
- Extended Computational Facilities
- User-Defined Collating Sequences
- Eased Programming Rules
- COBOL Source Program Debug Language

Each of these features is separately described on the following pages.

In addition, the Compiler accepts source language in support of the complete 1968 Standard—including the Report Writer.

IBM extensions (such as passwords for file security) are also supported.

VIRTUAL SYSTEM SUPPORT

OS/VS COBOL makes available the advantages of the OS/VS1 and OS/VS2 systems, the high-performance VSAM access method, and (with OS/VS2) the advantages of TSO—including the ability to use the TSO COBOL Prompter and OS COBOL Interactive Debug Program Products. In addition, powerful System/370 instructions are generated, and the performance of the advanced System/370 devices exploited. OS/VS1 and OS/VS2 can operate as independent systems, or under control of VM/370.

OS/VS COBOL also operates, with restrictions, under control of the CMS component of VM/370 (see the CMS Compatibility section for further information).

ENHANCED VSAM SUPPORT

VSAM is a high-performance access method of OS/VS1 and OS/VS2 for use with direct access storage. VSAM offers high-speed retrieval and storage of data, flexible data organization, ease of use—including simplified job control statements, data protection against unauthorized access, central control of data management functions, cross-system compatibility, device independence (freedom from consideration of block sizes, control information, record deblocking, etc.), the ability to monitor the execution status of each input/output request, and ease of conversion from other access methods. COBOL supports indexed files with alternate indexes (through VSAM KSDS capabilities), sequential files (through VSAM ESDS capabilities), and relative files (through VSAM RRDS capabilities). VSAM provides a multifunction utility program known as Access Method Services to define a VSAM data set, to load records into it, to convert an existing indexed or sequential data set to VSAM format, and to perform other tasks as well. VSAM allows key-sequenced, entry-sequenced, and relative record data sets.

In a key-sequenced data set (KSDS), records are stored in the ascending collating sequence of an embedded prime record key field. Records can be retrieved sequentially in prime key sequence or alternate key sequence; they can also be retrieved randomly according to the value of the desired key. VSAM uses the contents of the key field and optional free space (space in the data set not occupied by data) to insert new records in their key sequence. Using dynamic access, the programmer can specify sequential and/or random processing. More than one record in a file may have the same value for the record field associated with the alternate key by specifying the DUPLICATES phrase.

In an entry-sequenced data set (ESDS), the records are stored in the order in which they are presented for inclusion in the data set. New records are stored at the end of the data set. In COBOL, record retrieval must be sequential.

In a relative record data set (RRDS), the records are stored in ascending order of relative record numbers, and this data set provides the capability to access a mass storage file sequentially or randomly. Each record in a relative file is uniquely identified by an integer number representing the record's logical ordinal position in the file.

INDEXED FILE PROCESSING: A VSAM indexed (key-sequenced) file is ordered in the ascending sequence of its prime record key, which is embedded in the record and whose value must not change. Embedded alternate record keys can be specified for retrieval in another sequence. Records can be processed sequentially and/or randomly, and can be fixed or variable in length. Files can be extended.

In sequential access, records are accessed in the ascending order of their record key values or alternate record key values.

In random access, the sequence of record retrieval is controlled by the programmer. The desired record is accessed through the value placed in its RECORD KEY or ALTERNATE RECORD KEY data item. A full or partial key value can be used.

In dynamic access, records are processed sequentially or randomly, depending on the specific input/output request.

Performance Considerations: In a VSAM data set, inserted records are stored and addressed the same way as original records; thus, access speed after many insertions is equivalent to access speed before insertions. Free space allows efficient automatic reorganization of the data set by the access method; thus, there is seldom need to reorganize the data set offline.

SEQUENTIAL FILE PROCESSING: In a VSAM sequential (entry-sequencded) file, data is stored in the order in which it is received. In COBOL, records can be retrieved only in the same order in which they were stored. There is no key field when the file is extended. New records are always stored at the end of the data set. Records can be fixed or variable in length.

Programming Considerations: A record cannot be deleted from the file; however, its space can be reused for a record of the same length. If file reorganization becomes necessary, then a new file must be created, using records from the existing file.

RELATIVE RECORD FILE PROCESSING: In VSAM relative files (VSAM RRDS), the records are stored and retrieved in the order of their relative record numbers. Storage and retrieval can be sequential, random, or dynamic.

EXPANDED PHYSICAL SEQUENTIAL FILE CAPABILITIES

With the OPEN EXTEND option, the user is able to open the file for output operations. When an OPEN EXTEND statement is executed, the file is prepared for the addition of records immediately following the last record in the file.

Through the FILE STATUS clause, a value is placed into the specified two-character data item to indicate the status of a given input/output operation.

The LINAGE clause specifies the depth of a logical page as a line number, and, optionally, specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page.

ADDED COMMUNICATION SUPPORT

The Communication feature allows the COBOL programmer to access, process, and create messages and portions of messages, and to control the flow of messages through a communication network. Communication with local and remote communications devices is through a Message Control System (MCS).

The Communication feature also allows the OS/VS COBOL user to create device-independent message processing programs for communication applications.

In Communication applications, data flow into the system is random and at relatively low speeds. Data in the system exists as messages from remote stations, or as messages generated by internal programs. Once delivered to the system, however, messages can be processed at computer speeds.

The MCS acts as the logical interface between the entire network of communications devices and the COBOL program, in much the same manner as the operating system acts as an interface between the COBOL object program and conventional input/output devices. The MCS also performs device-dependent tasks, such as character translation from terminal code to and from computer code (EBCDIC), and insertion of control characters. Thus, the COBOL Communications program is device-independent.

The MCS has two constituent parts: A user-written Telecommunications Access Method (TCAM) Message Control Program (MCP) coded in assembly language, and COBOL communications feature object-time subroutines.

To store messages until they can be processed, the MCS uses message queues, which are similar to sequential data sets. The queues act as buffers for both the COBOL Communications program and the remote stations. That is, the COBOL Communications program accepts messages from MCS queues, and places messages into an MCS queue as if the queues were sequential files in a conventional COBOL program. To the COBOL program, the MCS queue from which it accepts messages is logically an input queue; the queue into which it places messages is logically an output queue. In this discussion, these terms are used with these meanings. A COBOL program need not accept or transmit complete messages from/to the MCS; portions of messages, known as message segments, may also be processed.

A message can be logically subdivided into message segments, which are delimited by End of Segment Indicators (ESI). A message is delimited from the next message by an End of Message Indicator (EMI). A group of messages can also be delimited from another group by an End of Group Indicator (EGI). The presence of these logical indicators is recognized and specified both by the MCS and the COBOL program; however, no indicators are included in the message text processed by COBOL programs. For incoming messages, the associated end indicators are identified in the CD entry area. For outgoing messages, the COBOL program specifies the end indicators to be associated with the message.

The interface between COBOL and the MCS is established through the Communication Description (CD) entries in the Communication Section. If input communication operations are to be performed, there must be at least one CD entry for input; if output communication operations are to be performed, there must be at least one CD entry for output. Multiple input and/or output CD entries are allowed.

Each CD entry is an implicitly defined fixed storage area into which information about messages is placed for use by both the COBOL program and the MCS.

The following five statements are used by the COBOL object program in the Procedure Division to request MCS services:

> ENABLE allows data transfer betwwen the MCS and the communication network.
>
> DISABLE prevents data transfer between the MCS and the communication network.
>
> RECEIVE causes data in an MCS input queue associated with a specified queue structure to be passed to the COBOL object program.
>
> SEND causes data associated with the COBOL object program to be passed to one or more MCS output queues.
>
> ACCEPT MESSAGE COUNT causes the MCS to return to the COBOL object program the number of complete messages in the MCS input queues associated with the specified queue structure.

A COBOL Communication program may be scheduled for execution through job control language. It may also be scheduled for execution by the MCS.

Figure 1 on page 10 illustrates the COBOL Communication environment.

Programming Considerations: For input, names identifying queues in the input CD entry must be equated through the DD statement to names used in the MCS. For output, the names in the output CD entry that identify symbolic destinations must be known to the MCS. The system provides a single 200-character buffer for use by all queues; the buffer size can be increased or decreased. This does not restrict the amount of data the COBOL program can request.

The user can test a COBOL Communication program by using physical sequential data sets and linking to BSAM instead of to TCAM. In general, this is accomplished through a JCL change only; the COBOL program need not be changed.

EXPANDED LIBRARY FACILITIES

The COPY statement provides a COBOL user with the ability to insert prewritten COBOL entries, which reside in a library or libraries, into a COBOL source program at compile time.

Library text associated with a text-name is copied into the source program, logically replacing the entire COPY statement beginning with the word COPY and ending with the period. When the REPLACING option is not specified, the library text is copied unchanged. Replacement text can be a literal, a word, or any COBOL character string(s).

A COPY statement may appear in the source program anywhere a character string or a separator may appear; however, a COPY statement must not be specified within a COPY statement.

POWERFUL DATA MANIPULATION

The INSPECT statement provides the user the ability to specify that a character, or group(s) of characters in a data item are to be counted, replaced, or counted and replaced. INSPECT dynamically determines the length of a data item and initializes a data item to zeros or spaces partially or completely.

STRING and UNSTRING statements give the OS/VS COBOL user greater flexibility in data manipulation. STRING gives the programmer the ability to put together the partial or complete contents of two or more data items in one single data item; UNSTRING causes contiguous data in a sending field to be separated and placed into multiple receiving fields. In either case, the sending field(s) can be specified as containing one or more delimiters—characters which terminate the data transfer for the field, but are not themselves transferred. Delimiters can be specified at compile time or at object time. If the execution of the statement completely fills the receiving field(s) before the end of the sending field(s) is reached, an overflow condition exists; provision is made for an ON OVERFLOW exit.

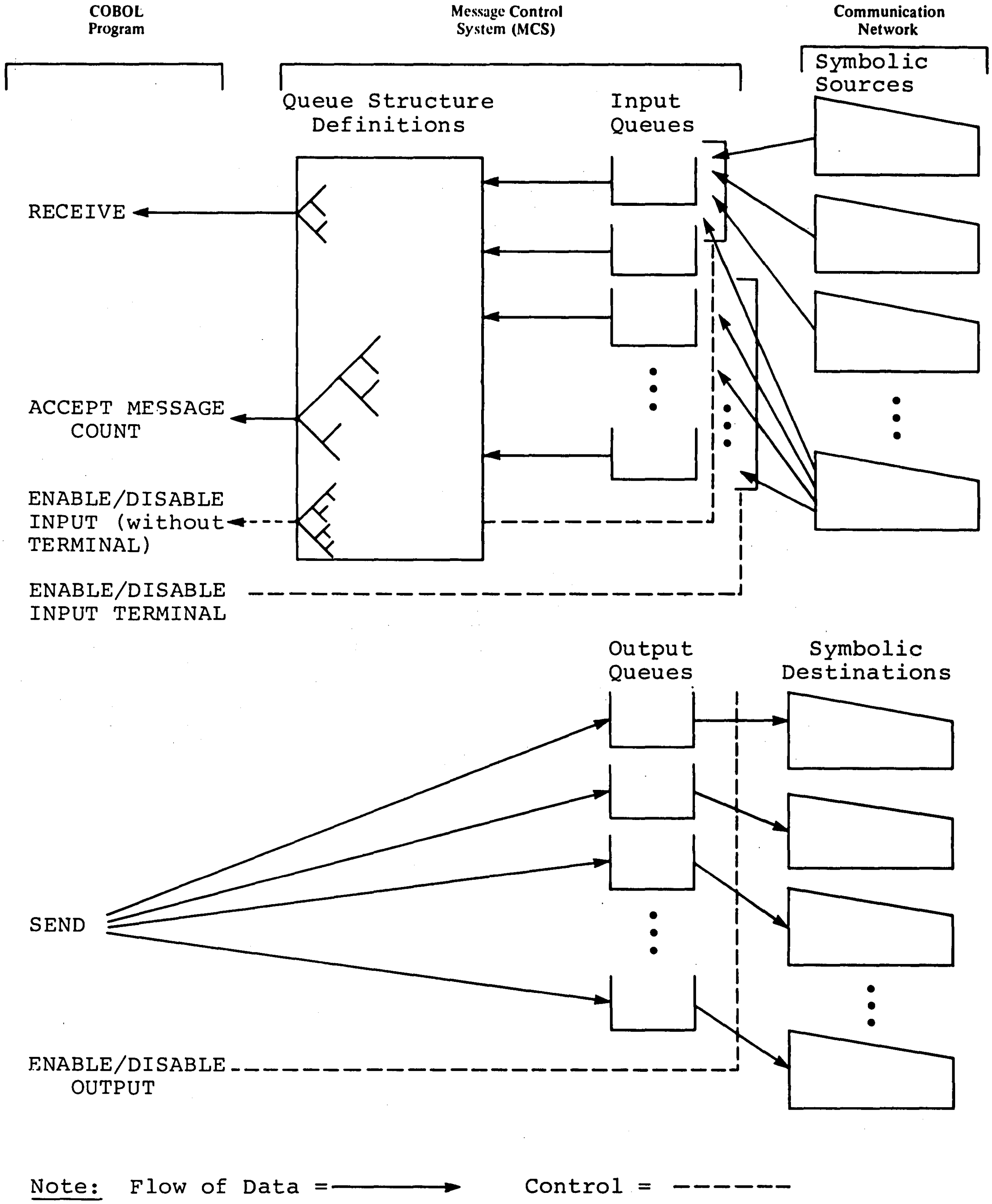

COBOL Program
Message Control System (MCS)
Communication Network
Symbolic Sources
Queue Structure Definitions
Input Queues
RECEIVE
ACCEPT MESSAGE COUNT
ENABLE/DISABLE INPUT (without TERMINAL)
ENABLE/DISABLE INPUT TERMINAL
Output Queues
Symbolic Destinations
SEND
ENABLE/DISABLE OUTPUT
Note: Flow of Data = ——→ Control = - - - - - -

Figure 1. COBOL Communication Environment

EXTENDED COMPUTATIONAL FACILITIES

When the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation.

The ADD, SUBTRACT, MULTIPLY, and DIVIDE arithmetic statements are used for computations and can be coded with GIVING and a series of identifiers. These arithmetic statements allow multiple receiving fields.

The COMPUTE statement allows the user to assign one or more data items the value of an arithmetic expression. COMPUTE allows the user to combine arithmetic operations without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

USER-DEFINED COLLATING SEQUENCES

The User-Defined Collating Sequences can be specified for an entire program, for physical sequential data files, or for Sort/Merge comparison.

The COLLATING SEQUENCE clause allows the user to specify any number of alphabets in the Environment Division, any one of which may then be used as the Program Collating Sequence (for nonnumeric compares). Also, a different SORT/MERGE collating sequence may be specified for each SORT or MERGE statement. The file description statement allows a CODE-SET clause which names an alphabet defined as NATIVE (EBCDIC) or STANDARD-1 (ASCII); but not user defined.

MERGE FACILITY

The MERGE verb allows the COBOL user to combine two or more identically ordered input files into one output file according to embedded key(s) in the record. The MERGE verb uses the IBM Program Product OS/VS Sort/Merge (Program Number 5740-SM1). Special processing of merged records can also be specified. More than one MERGE statement can be executed in one program. Both standard sequential files and sequentially accessed VSAM files can be designated as input or output.

SYSTEM/370 DEVICE SUPPORT

In OS/VS COBOL, the actual device upon which a file resides is resolved at object time through JCL. This means that any valid System/370 OS/VS device whose functions correspond to COBOL language capabilities can be used by the COBOL program. Such device specification, in most cases, is not apparent to the COBOL program. However, special considerations apply to the following:

3886 Optical Character Reader: This general purpose online device satisfies a broad range of data entry requirements. The device scans documents line-by-line and the contents are transmitted line-by-line to the processor. When the entire document has been processed, it is ejected.

OS/VS COBOL supports the IBM 3886 Optical Character Reader through object-time library subroutines. Functions provided include: opening and closing the file, reading, checking, controlling the document, and loading a new format record. After each I/O request, a code is returned to the program so that any exceptional condition can be detected. The OS/VS COBOL library contains the object-time subroutine.

System/370 Multifunction Card Devices: OS/VS COBOL supports the combined function processing available through these devices. Combined functions available are: read/punch, read/print, read/punch/print, and punch/print. The functions must be performed in the order shown. The print function may use the AFTER ADVANCING or AFTER POSITIONING options of the WRITE statement.

All devices supported by OS Full American National Standard COBOL Version 4 are supported by OS/VS COBOL. These include, among others: IBM 3504/3505 Card Reader (with OMR/RCE) and IBM 3410/3420 Magnetic Tape Units.

DYNAMIC SUBPROGRAM LINKAGE

Through the CALL and CANCEL statements, OS/VS COBOL allows static or dynamic loading and deletion of subprograms. Through the PARM field of the EXEC job control statement invoking the compiler, the user can specify the mode (static or dynamic) of the CALL literal statement. The CALL identifier statement is always dynamic.

When the CALL statement is static, the main COBOL program and all invoked subprograms must be part of the same load module. Thus, when a subprogram is called it is already resident in storage, and a branch to it occurs. When subsequent CALL statements are executed, the subprogram is entered in its last-used state. If alternate entry points are specified, then any CALL statement to the subprogram can select any of the alternate entry points at which to enter the subprogram.

When the CALL statement is dynamic, the COBOL user can control the loading of subprograms; that is, the called subprogram is not link edited with the main program. At execution time, the subprogram is loaded only if and when it is called.

Each subprogram invoked with a dynamic CALL statement may be part of a different load module, which is a member of the system link library or of a user-supplied private library. The execution of the dynamic CALL statement to a subprogram that is not currently resident in storage results in the loading of that subprogram from secondary storage into the region/partition containing the main program, and a branch to the subprogram.

Thus, the first dynamic CALL to a subprogram obtains a fresh copy of the subprogram. Subsequent calls to the same subprogram (either by the original caller or by any other subprogram within the same region or partition) result in a branch to the same copy of the subprogram in its last-used state. If the ON OVERFLOW option is specified, and there is not enough storage available to accommodate the called subprogram, the ON OVERFLOW imperative-statement is executed.

However, when a CANCEL statement is issued for that subprogram, the storage occupied by the subprogram is freed, and a subsequent CALL acts as though it were the first. A CANCEL statement referring to a called subprogram may be issued by a program other than the original caller.

When dynamic subprogram linkage is used, the COBOL Library Management Facility must also be used by the main program and all subprograms in one region/partition. Otherwise, multiple copies of library subroutines may be resident at one time and cause unpredictable results.

User subprograms that are to be invoked at object time with the dynamic CALL statement must be members of the system link library or of a user-supplied private library. For the CALL statement to function as defined by the 1974 Standard, user subprograms must be link-edited as nonreentrant and nonserially-reusable.

PROGRAM DEVELOPMENT AIDS

EASED PROGRAMMING RULES

Table Handling allows literal subscripts to be mixed with index-names when referring to a table item. Negative literals may be used in the SET statement. The SET statement establishes reference points for table handling operations by placing values associated with table elements into indexes associated with index-names.

Level 77 items in the Data Division under the Working-Storage Section can follow level 01 items.

Relaxed punctuation rules allow a space before a period, comma, or semicolon, and a space may immediately precede or may immediately follow a parenthesis (except in PICTURE).

Also, a slash in column 7 causes that line to be treated as a comment, and starts the comment on a new page.

COBOL SOURCE PROGRAM DEBUG LANGUAGE

The Debugging features allow the user to specify conditions under which data items or procedures are to be monitored during program execution.

COBOL language elements that implement the Debugging features are: a compile-time switch (WITH DEBUGGING MODE), an object-time switch, USE FOR DEBUGGING Declarative, a special register DEBUG-ITEM, and Debugging lines (which can be written anywhere after the Object Computer paragraph in the Environment, Data, and Procedure Divisions).

Compile-Time Switch—in the Source-Computer paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.

The WITH DEBUGGING MODE clause indicates that all Debugging sections and all Debugging lines are to be compiled. If this clause is not specified, all Debugging lines and sections are compiled as if they were comment lines.

Object-Time Switch—dynamically activates the Debugging code generated when WITH DEBUGGING MODE is specified.

The USE FOR DEBUGGING Declarative identifies the items in the source program that are to be monitored by the associated declarative procedure.

The DEBUG-ITEM special register provides information about the conditions causing Debugging section execution, and can be referenced only in a Debugging Declarative.

A Debugging line is any line in a source program with a D coded in column 7, the continuation area.

FEDERAL INFORMATION PROCESSING STANDARD (FIPS) FLAGGER

The 1975 Federal Information Processing Standard (FIPS) is a compatible subset of American National Standard COBOL, X3.23-1974, and the 1972 Federal Information Processing Standard (FIPS) is a compatible subset of American National Standard COBOL, X3.23-1968. FIPS recognizes four language levels: low, low-intermediate, high-intermediate, and full. When the FIPS Flagger is used, source clauses and statements that do not conform to FIPS are identified. This assists the user in creating OS/VS COBOL programs which conform to the specified level.

The Federal Information Processing Standard (FIPS) COBOL flagger issues messages identifying nonstandard elements in a COBOL source program. The FIPS flagger makes it possible to ensure that COBOL clauses and statements in an OS/VS COBOL source program conform to the specified level of either the 1975 or the 1972 Federal Information Processing Standard COBOL.

Programming Considerations: At installation time, no flagging, NOLVL (which is the system generation default), or flagging at a specified FIPS level, LVL=A/B/C/D, can be specified as the installation default option. At compile time, the programmer can override any of these options by specifying another level of FIPS flagging; if NOLVL is specified, however, the option is ignored and the default LVL option is used. Through the LANGLVL option, the programmer can specify flagging for either the 1972 FIPS or the 1975 FIPS.

MIGRATION FLAGGING

Migration flagging analyzes COBOL source programs and issues informational messages indicating COBOL statements that are no longer supported or that are supported differently by the VS COBOL II Compiler, Program Number 5668-958. This helps programmers migrate programs from OS/VS COBOL to VS COBOL II.

For information about VS COBOL II, see VS COBOL II General Information.

INTERACTIVE CAPABILITIES

The Compiler can provide output directed to a TSO or CMS terminal. The terminal user can determine the characteristics of compiler output to the terminal, such as:

- Compilation progress, diagnostic messages and compiler diagnostics.

  The TERM option orders the compiler, as it processes the source program, to issue a progress message, and to issue compiler error and warning messages to the terminal—each compiler message including the line number of the source statement in error and the message text. After all diagnostic messages are issued, a message stating the total number of statements in error is produced at the terminal. Using Edit Mode subcommands, the programmer can retrieve the statement by line number and correct the error before recompiling.

- The compiler's entire listing data set.

  The PRINT/NOPRINT option allows the programmer to choose whether the program listing is to be placed in a data set, displayed at the terminal, or suppressed. If the NOPRINT option is specified, the compiler does not allocate a SYSPRINT file. This saves compilation time and storage.

Through the NUM option, user-recorded line numbers in the input data set may be substituted for internal statement numbers in any diagnostic messages printed on the terminal.

Display and debugging output from a COBOL program can be directed to the terminal, and the ACCEPT statement can invoke input from the terminal.

Note: Additional interative capabilities are provided through the separate Program Products TSO COBOL Prompter and COBOL Interactive Debug. Both Program Products are described in the section on Related COBOL Development Aids.

WHEN-COMPILED SPECIAL REGISTER

The WHEN-COMPILED special register makes available to the object program the date-and-time compiled constant carried in the object module. WHEN-COMPILED provides a means of associating a compilation listing with both the object program and the output produced at execution time.

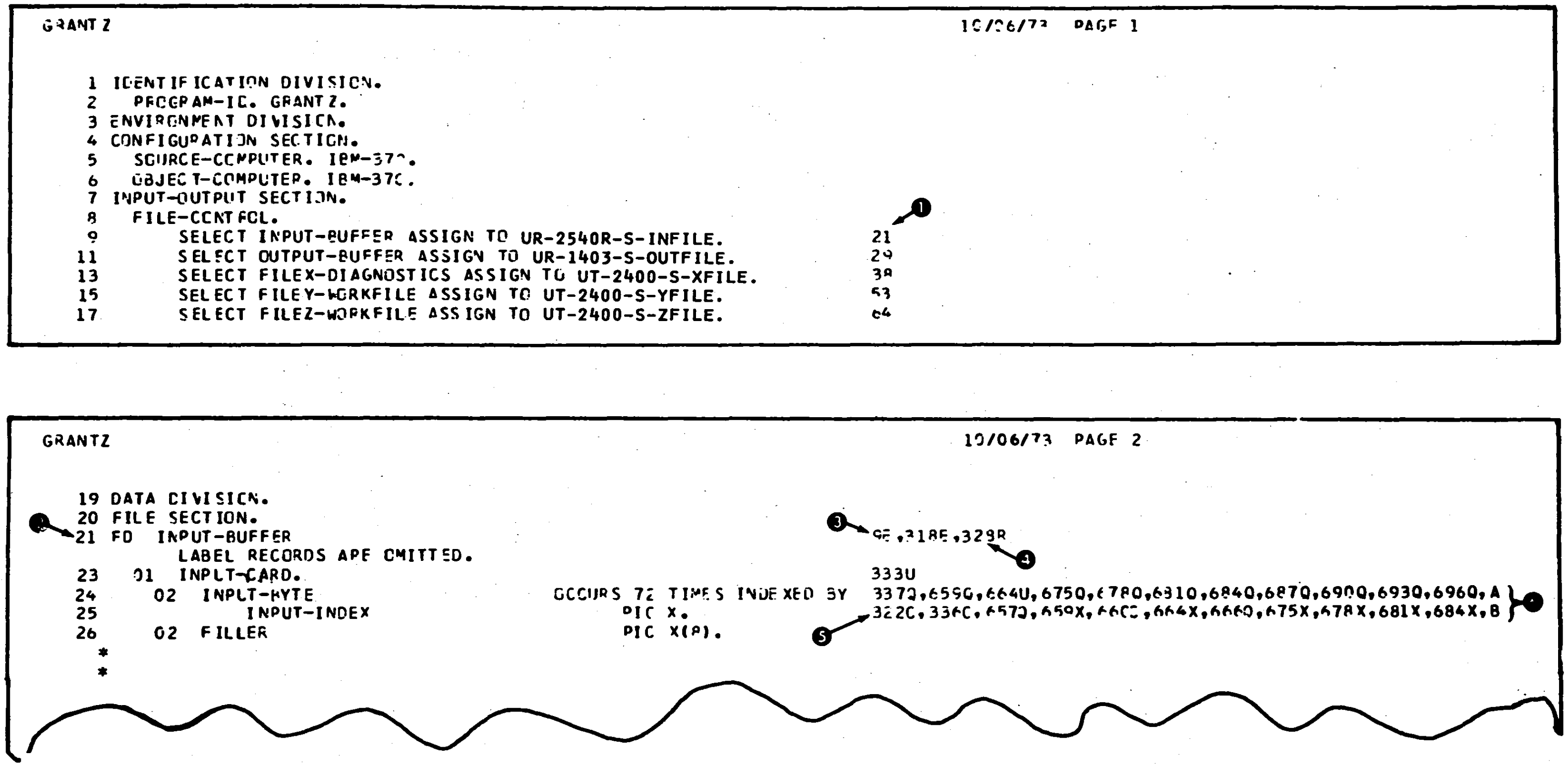
LISTER FACILITY

The Lister facility permits the user to obtain a reformatted detailed source code listing that contains complete cross-reference information at the source level. Each COBOL statement is begun on a new line, and indented in a manner that makes the logic of the program readily apparent, by highlighting level numbers, nested IF statements, etc. Each line of the reformatted source listing contains one or more references to other source statements, specifying the statement number, and indicating the type of reference.

Thus, when reading the Data Division, the user can identify immediately the Procedure Division statements that read, write, or inspect a given data item. File descriptions are comprehended quickly because of uniform indenting conventions that are imposed by the Lister facility. When reading the Procedure Division, the user can see references to statements in the Data division, showing use of the data item, and also to other statements in the Procedure Division, simplifying the tracing of program logic. The Lister facility further facilitates the tracing of program logic by optionally printing the Procedure Division in 2-column format, so that fewer page turnings are required, and more logic appears on a page. Optionally, the user may also obtain a new source deck that reflects the reformatted source listing, with the exception of embedded cross-reference information. Figure 2 on page 16 gives an example of Lister output.

The Lister facility also produces a summary listing that contains selected statements from the source program, plus a condensation of the detailed information from the reformatted source listing. The total number of each type of reference for each File Description, and for each Section in the Procedure Division is shown.

Performance Considerations: Although use of the Lister facility necessarily adds time to the compilation run, the Lister facility does not alter the source code. Therefore, such source code takes no longer to compile than if the Lister option had not been invoked. However, to save time on initial compilations, the Lister facility should not be invoked until the COBOL source program is known to be substantially free of syntax errors. Once an up-to-date reformatted source listing and new source deck reflecting the listing is obtained, the Lister facility can be omitted on subsequent compilations. When large COBOL programs are to be listed and compiled, the user may be able to obtain shorter run times by electing to use Lister cross-reference information in place of XREF or SXREF.



```
GRANT Z                                                                              1C/C6/73   PAGF 1


     1 ICENTIFICATION DIVISICN.
     2   PRCGRAM-IC. GRANTZ.
     3 ENVIRCNMENT DIVISICN.
     4 CONFIGURATION SECTICN.
     5   SOURCE-CCMPUTER. IBM-37^.
     6   OBJECT-COMPUTER. IBM-37C.
     7 INPUT-OUTPUT SECTION.
     8   FILE-CCNTROL.
     9        SELECT INPUT-BUFFER ASSIGN TO UR-2540R-S-INFILE.                21
    11        SELECT OUTPUT-BUFFER ASSIGN TO UR-1403-S-OUTFILE.               29
    13        SELECT FILEX-DIAGNOSTICS ASSIGN TO UT-2400-S-XFILE.             38
    15        SELECT FILEY-WORKFILE ASSIGN TO UT-2400-S-YFILE.                53
    17        SELECT FILEZ-WORKFILE ASSIGN TO UT-2400-S-ZFILE.                64
```

```
GRANTZ                                                                               10/06/73   PAGF 2


    19 DATA CIVISICN.
    20 FILE SECTION.
    21 FD  INPUT-BUFFER                                                       9E,318E,329R
            LABEL RECORDS APE CMITTED.
    23   01  INPUT-CARD.                                                      333U
    24     02  INPUT-BYTE                        OCCURS 72 TIMES INDEXED BY   3370,6590,664U,6750,6780,6810,6840,6870,6900,6930,6960,A
    25               INPUT-INDEX                           PIC X.             322C,336C,6570,659X,660C,664X,6660,675X,678X,681X,684X,B
    26     02  FILLER                                      PIC X(8).
       *
       *
```

1 Reference to FD statement number.

2 FD referred to by SELECT statement.

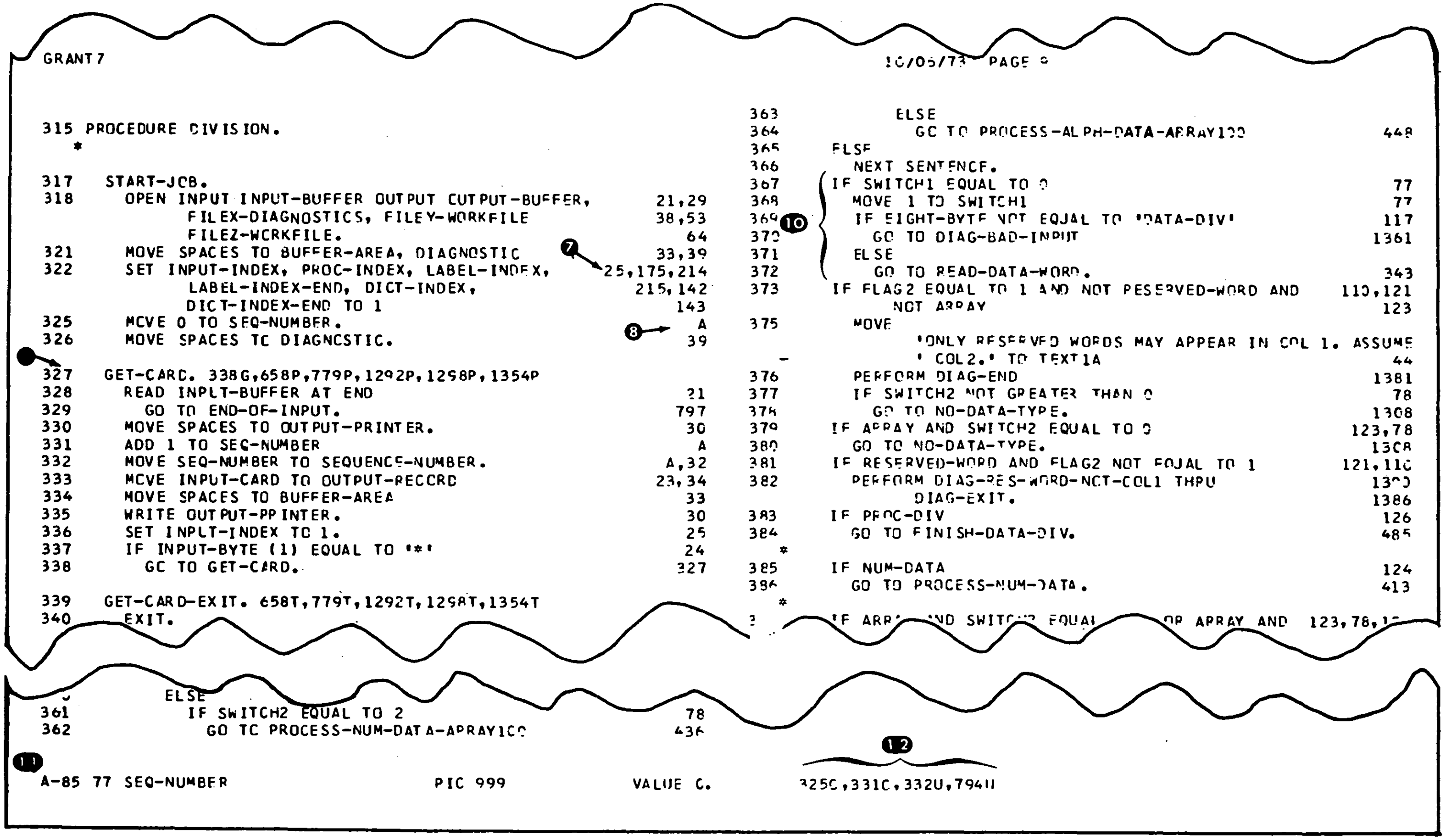
3 SELECT statement; (E) denotes Environment Division reference.

4 Procedure Division statement; (R) denotes that statement 328 reads this data item.

5 Data item changed by statement 322.

6 Footnotes for additional references at bottom of this page.

Figure 2 (Part 1 of 2). Example of Lister Facility Output



```
 GRANT7                                                                        1C/05/73  PAGE 9

                                                                 363           ELSE
 315 PROCEDURE CIVISION.                                         364             GC TO PROCESS-ALPH-DATA-ARRAY100                448
     *                                                           365       ELSE
                                                                 366         NEXT SENTENCE.
 317    START-JCB.                                               367       IF SWITCH1 EQUAL TO 0                                  77
 318      OPEN INPUT INPUT-BUFFER OUTPUT CUTPUT-BUFFER,    21,29  368         MOVE 1 TO SWITCH1                                    77
                FILEX-DIAGNOSTICS, FILEY-WORKFILE          38,53  369         IF EIGHT-BYTE NOT EQJAL TO 'DATA-DIV'               117
                FILEZ-WCRKFILE.                               64  370           GO TO DIAG-BAD-INPUT                              1361
 321      MOVE SPACES TO BUFFER-AREA, DIAGNOSTIC           33,39  371         ELSE
 322      SET INPUT-INDEX, PROC-INDEX, LABEL-INDEX,   25,175,214  372           GO TO READ-DATA-WORD.                              343
                LABEL-INDEX-END, DICT-INDEX,             215,142  373       IF FLAG2 EQUAL TO 1 AND NOT PESERVED-WORD AND      110,121
                DICT-INDEX-END TO 1                          143                 NOT ARRAY                                       123
 325      MCVE 0 TO SEQ-NUMBER.                                A  375         MOVE
 326      MOVE SPACES TC DIAGNCSTIC.                          39                   'ONLY RESERVED WORDS MAY APPEAR IN COL 1. ASSUME
                                                                   -               ' COL2.' TO TEXT1A                                 44
 327    GET-CARD. 338G,658P,779P,1292P,1298P,1354P                376         PERFORM DIAG-END                                   1381
 328      READ INPLT-BUFFER AT END                            21  377         IF SWITCH2 NOT GREATER THAN 0                       78
 329        GO TO END-OF-INPUT.                              797  378           GO TO NO-DATA-TYPE.                              1308
 330      MOVE SPACES TO OUTPUT-PRINTER.                      30  379       IF ARRAY AND SWITCH2 EQUAL TO 0                    123,78
 331      ADD 1 TO SEQ-NUMBER                                  A  380         GO TO NO-DATA-TYPE.                                1308
 332      MOVE SEQ-NUMBER TO SEQUENCE-NUMBER.               A,32  381       IF RESERVED-WORD AND FLAG2 NOT EOJAL TO 1          121,110
 333      MCVE INPUT-CARD TO OUTPUT-RECCRD                 23,34  382         PERFORM DIAG-RES-WORD-NCT-COL1 THRU                1300
 334      MOVE SPACES TO BUFFER-AREA                          33                     DIAG-EXIT.                                 1386
 335      WRITE OUTPUT-PRINTER.                               30  383       IF PROC-DIV                                          126
 336      SET INPLT-INDEX TC 1.                               25  384         GO TO FINISH-DATA-DIV.                              485
 337      IF INPUT-BYTE (1) EQUAL TO '*'                      24       *
 338        GC TO GET-CARD.                                  327  385       IF NUM-DATA                                          124
                                                                 386         GO TO PROCESS-NUM-DATA.                            413
 339    GET-CARD-EXIT. 658T,779T,1292T,1298T,1354T                     *
 340      EXIT.                                                    3       IF ARR   ND SWITC  EQUAL      OR ARRAY AND  123,78,1

               ELSE
 361             IF SWITCH2 EQUAL TO 2                        78
 362               GO TC PROCESS-NUM-DATA-ARRAY1CO           436

 A-85 77 SEQ-NUMBER                        PIC 999                 VALUE C.        325C,331C,332U,794U
```

Figure 2 (Part 2 of 2). Example of Lister Facility Output

Note two-column Procedure Division listing.

7 Data items.

8 Footnoted data item.

9 Paragraph Gone to (G) and Performed (P) by referenced statements.

10 Nested IFs indented progressively.

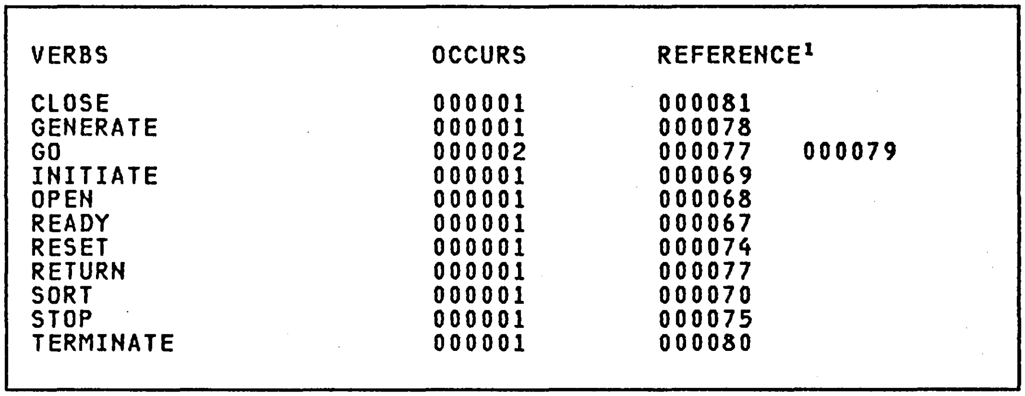
11 Footnote.

12 Procedure Division statements referencing this data item.

VERB PROFILES

The verb profiles option produces a list of all verbs contained in the COBOL source program. Each different COBOL verb in the program is shown, followed by a count representing the number of times it appears in the source program. Optionally, the count is followed by the associated statement numbers.

Figure 3 gives an example of verb profile output.



```
VERBS                    OCCURS        REFERENCE[1]

CLOSE                    000001        000081
GENERATE                 000001        000078
GO                       000002        000077    000079
INITIATE                 000001        000069
OPEN                     000001        000068
READY                    000001        000067
RESET                    000001        000074
RETURN                   000001        000077
SORT                     000001        000070
STOP                     000001        000075
TERMINATE                000001        000080
```

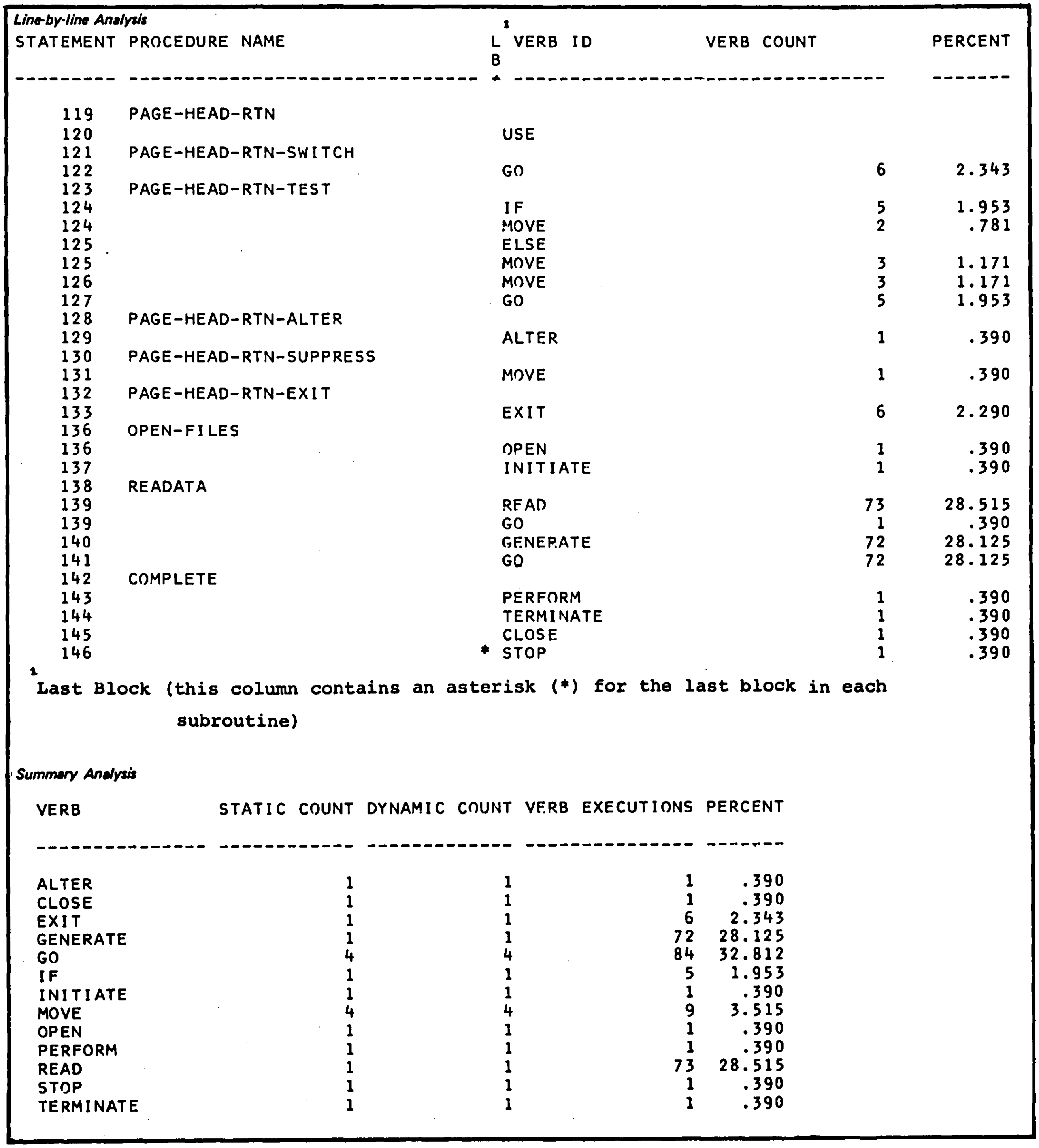
Figure 3. Example of Verb Profile Output

Note

[1] Statement number references can be requested optionally in addition to the verb count.

EXECUTION-TIME STATISTICS

The execution-time statistics option provides the programmer with information that aids user program optimization by identifying heavily-used portions of the COBOL source program. It is also useful to the programmer in debugging by providing verification that all portions of a program have been executed. During execution of the object program, a count is maintained for each verb in the source program. Just prior to program termination, the system prints the accumulated execution count showing the verb, the name of the procedure in which it appears, and the number of the statement in which it appears. Figure 4 on page 19 gives an example of Execution-Time Statistics output.



Line-by-line Analysis

```
                                                       1
STATEMENT PROCEDURE NAME                               L VERB ID          VERB COUNT          PERCENT
                                                       B
--------- ------------------------------ * ---------------------------------   -------

    119   PAGE-HEAD-RTN
    120                                      USE
    121   PAGE-HEAD-RTN-SWITCH
    122                                      GO                                 6      2.343
    123   PAGE-HEAD-RTN-TEST
    124                                      IF                                 5      1.953
    124                                      MOVE                               2       .781
    125                                      ELSE
    125                                      MOVE                               3      1.171
    126                                      MOVE                               3      1.171
    127                                      GO                                 5      1.953
    128   PAGE-HEAD-RTN-ALTER
    129                                      ALTER                              1       .390
    130   PAGE-HEAD-RTN-SUPPRESS
    131                                      MOVE                               1       .390
    132   PAGE-HEAD-RTN-EXIT
    133                                      EXIT                               6      2.290
    136   OPEN-FILES
    136                                      OPEN                               1       .390
    137                                      INITIATE                           1       .390
    138   READATA
    139                                      READ                              73     28.515
    139                                      GO                                 1       .390
    140                                      GENERATE                          72     28.125
    141                                      GO                                72     28.125
    142   COMPLETE
    143                                      PERFORM                            1       .390
    144                                      TERMINATE                          1       .390
    145                                      CLOSE                              1       .390
    146                                    * STOP                               1       .390
```

[1] Last Block (this column contains an asterisk (*) for the last block in each subroutine)

Summary Analysis

```
VERB            STATIC COUNT DYNAMIC COUNT VERB EXECUTIONS PERCENT

--------------- ------------ ------------- --------------- -------

ALTER                      1             1               1    .390
CLOSE                      1             1               1    .390
EXIT                       1             1               6   2.343
GENERATE                   1             1              72  28.125
GO                         4             4              84  32.812
IF                         1             1               5   1.953
INITIATE                   1             1               1    .390
MOVE                       4             4               9   3.515
OPEN                       1             1               1    .390
PERFORM                    1             1               1    .390
READ                       1             1              73  28.515
STOP                       1             1               1    .390
TERMINATE                  1             1               1    .390
```

Figure 4. Example of Execution-Time Statistics Output

BACKGROUND SYMBOLIC DEBUG

This feature reduces program development time by making debugging information available in a COBOL format instead of a hexadecimal dump format. No source language changes are needed; the debugging information is requested through object-time control cards. Thus, multiple debugging runs can be made without recompilation.

When a program terminates abnormally, the user receives a COBOL-formatted map of his Data Division. Each data area is identified by its source name, and its contents are easily readable. The user can also request snapshots of the Data Division at any point during program execution.

If two or more COBOL programs are link-edited together and one of them terminates abnormally, the user is provided with such COBOL-formatted information for the program causing termination and its callers, up to and including the main program. Abnormal termination information is provided in the following parts:

1. Abnormal termination message—including the program-name, and the COBOL sequence number of the statement and of the verb being executed.

2. An Optional Flow Trace—if requested, a time-sequenced trace of the names of the last "n" COBOL procedures executed.

3. Selected areas in the Task Global Table.

4. COBOL-formatted map of the Data Division including:

   a. Working-Storage Section

   b. Linkage Section

   c. For FDs, the data record and file status summary

   d. For RDs, the report line, page counter, and line counter

   e. For SDs, the sort record

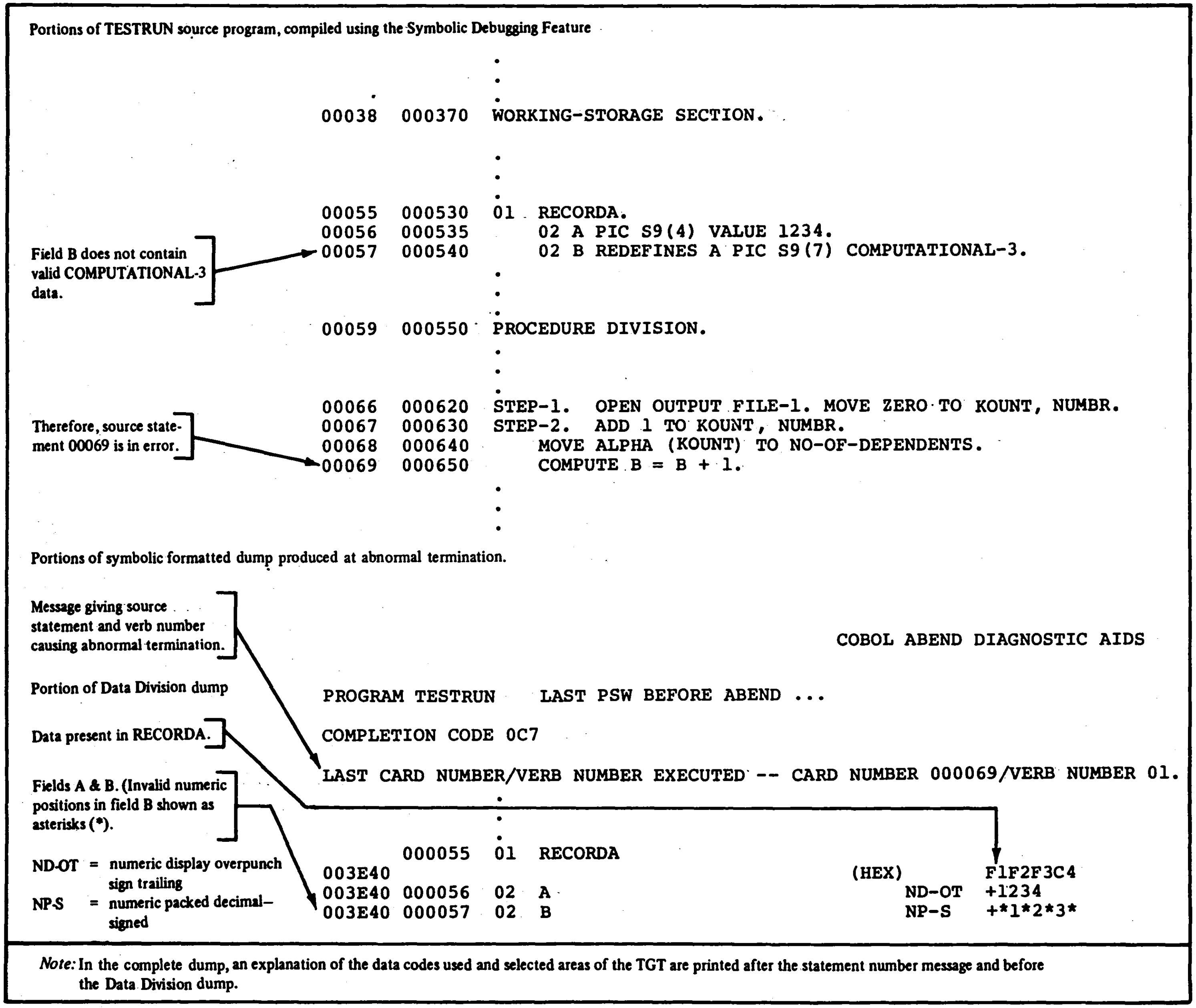
   f. For CDs, the CD itself in its implicit format, and the area containing the message data currently in the buffer

Figure 5 on page 21 gives portions of a sample program compiled using the Background Symbolic Debug feature, and shows an example of the abnormal termination information that can be requested.

At compile time, an option tells the compiler to create a debug file, an additional data set which contains descriptions of data items (the dictionary) and other debugging information. At object time, this debug file is required online, and symbolic debug output is requested through control cards.

Several COBOL programs link-edited together must have separate debug files if they each use the Background Symbolic Debug feature. This feature automatically provides optimized object code. The user can request that source sequence numbers be checked and corrected. When the user requests that dynamic snapshots be taken, they can be specified at a STOP RUN, EXIT PROGRAM, or GOBACK statement. In this case, an "end-of-job" snapshot results.

Note: The separate Program Product OS COBOL Interactive Debug is also available to the TSO or CMS COBOL user.



Portions of TESTRUN source program, compiled using the Symbolic Debugging Feature

```
                        .
                        .
                        .
00038  000370  WORKING-STORAGE SECTION.
                        .
                        .
                        .
00055  000530  01  RECORDA.
00056  000535      02 A PIC S9(4) VALUE 1234.
00057  000540      02 B REDEFINES A PIC S9(7) COMPUTATIONAL-3.
                        .
                        .
                        .
00059  000550  PROCEDURE DIVISION.
                        .
                        .
                        .
00066  000620  STEP-1.  OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT, NUMBR.
00067  000630  STEP-2.  ADD 1 TO KOUNT, NUMBR.
00068  000640      MOVE ALPHA (KOUNT) TO NO-OF-DEPENDENTS.
00069  000650      COMPUTE B = B + 1.
                        .
                        .
                        .
```

Field B does not contain valid COMPUTATIONAL-3 data.

Therefore, source statement 00069 is in error.

Portions of symbolic formatted dump produced at abnormal termination.

```
                                                   COBOL ABEND DIAGNOSTIC AIDS

PROGRAM TESTRUN    LAST PSW BEFORE ABEND ...

COMPLETION CODE 0C7

LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000069/VERB NUMBER 01.
                        .
                        .
                        .
        000055  01  RECORDA
003E40                                            (HEX)      F1F2F3C4
003E40  000056  02  A                                 ND-OT  +1234
003E40  000057  02  B                                 NP-S   +*1*2*3*
```

Message giving source statement and verb number causing abnormal termination.

Portion of Data Division dump

Data present in RECORDA.

Fields A & B. (Invalid numeric positions in field B shown as asterisks (*).

ND-OT = numeric display overpunch sign trailing

NP-S = numeric packed decimal–signed

Note: In the complete dump, an explanation of the data codes used and selected areas of the TGT are printed after the statement number message and before the Data Division dump.

Figure 5. Example of Background Symbolic Debugging Output

FLOW TRACE

The flow trace option allows the user to receive a formatted trace (that is, a list containing the program identification and statement numbers) corresponding to a variable number of procedures executed prior to an abnormal termination. The number of procedures to be traced is specified by the user. The flow trace option requires no source language changes.

A flow trace is printed only in the event of an abnormal termination. The number of procedures to be traced may be specified at compile time or execution time, and the output, when running under TSO, may be directed to the terminal.

SYNTAX-CHECKING COMPILATION

With the OS/VS COBOL Compiler, the user can request a syntax-checking compilation, through the PARM field of the EXEC control statement. Such compilations can be unconditional or conditional.

When unconditional syntax-checking is requested, the compiler scans the source text for syntax errors, and generates the appropriate error messages, but does not generate object text.

When conditional syntax-checking is requested, a full compilation is produced if no messages or only W- or C-level messages are generated; if one or more E-level or D-level messages are generated, no object code is produced.

Syntax-only compilation can considerably reduce compile time. Unconditional syntax checking can reduce compilation time more than conditional syntax checking.

Programming Considerations: When object text is not generated, the following compile-time options are suppressed: LOAD, XREF, SXREF, CLIST, NOSUPMAP, PMAP, DECK, VBSUM, VBREF, COUNT; if optimized object code is requested, the object code is not produced; if background symbolic debugging is requested, the symbolic debugging option is suppressed.

Unconditional syntax checking is assumed if all of the following compile-time options are specified:

NOLOAD	NOCLIST	SUPMAP
NODECK	NOMAP	NOXREF/NOSXREF

A full compilation—including error messages, object text (if requested), and all other specified (or default) options—is produced if:

- Neither unconditional nor conditional syntax-checking is specified
- Unconditional syntax-checking is not assumed

STATEMENT NUMBER OPTION

This option facilitates debugging by providing information about the statement being executed in the event of an abnormal termination. At abnormal termination, the statement number is printed; if there are two or more verbs in the source statement, the verb being executed is identified. The program containing the statement is also identified. This option requires no source language changes.

Performance Considerations: Five additional bytes are generated for each procedure-name in the program, and 5 to 17 additional bytes are generated for each verb.

EFFICIENT OBJECT-TIME PERFORMANCE

OPTIMIZED OBJECT CODE

The object code generated by the OS/VS COBOL Compiler can be optimized. When optimization is requested, the resulting object program contains fewer machine instructions than it would contain if optimization had not been requested; programs are divided into 4K-byte procedure blocks; register assignment is made more efficient than without optimization.

Use of the feature results in a considerable reduction in use of object program storage. The reduction in size is dependent upon source program size and content. In general, the larger the number of source statements, branching statements, and 01-level data names, the larger the percentage of reduction.

Programming Considerations: Optimization is requested at compile time through the OPT parameter in the PARM field of the EXEC job control statement.

Optimized object code is automatically provided when Background Symbolic Debug is specified; it may be requested when the flow trace or statement number options are requested.

COBOL LIBRARY MANAGEMENT FACILITY

The OS/VS COBOL library management facility allows a single copy of the COBOL library subroutines to be shared by all currently executing COBOL programs, even in different partitions or regions. (When the feature is not specified, all programs and subprograms, plus their required COBOL subroutines, are link-edited into one load module for execution in one partition/region. Thus, many copies of one COBOL subroutine may be resident in virtual (or real) storage at one time, one in each partition/region.)

When the library management facility is used, the COBOL library subroutines may be wholly or partially resident in the VS2 link pack area (LPA) or in the VS1 resident reusable routine area (RRR), or they may be resident within each partition/region. (Some routines cannot be so placed—such as the subroutine used for intraregion/intrapartition communication, the queue structure description routine, a STOP RUN routine, a special DISPLAY routine, etc.) The actual physical location of these routines is not apparent to the executing program.

The primary advantage in placing the COBOL library subroutines in the LPA/RRR area is the economy it allows in virtual storage allocation. Though the LPA/RRR area must be made larger to accommodate all the required COBOL library subroutines, it is pageable, and each region/partition no longer requires its own copy.

Programming Considerations: To be able to place the COBOL subroutines in the LPA/RRR area, the user must execute a utility program to add two members to the system parameter library. The members are:

1. A User List of all names and all aliases for those COBOL subroutines the user wishes to place in the LPA/RRR area.

2. A Linkage Routine that allows the concatenation of the system link library with the COBOL subroutine library, or with a private library containing selected COBOL subroutines. (Note that, if the user wishes to place selected COBOL subroutines into a private library, a utility program must be executed to catalog that library.)

At initial program loading (IPL) time, the user identifies the user list to the system. The system then uses the linkage routine to place the listed COBOL subroutines into the LPA/RRR area.

The COBOL library management facility is invoked at compile-time through the PARM field of the EXEC job control statement.

In any given region or partition, if the COBOL library management facility is used at all, it must be used by the main program and by all subprograms in that region or partition. Otherwise, multiple copies of COBOL library subroutines may be resident at the same time and cause unpredictable results.

For a region or partition not using the COBOL library management facility, the COBOL object program and the COBOL library subroutines it uses are link-edited together into one load module.

If COBOL library subroutines that were not loaded into the LPA/RRR area at IPL time are required for execution of the program, and the COBOL library management facility is being used, then:

- For a main program, such subroutines are loaded into the region/partition before execution of the main program.
- For a subprogram, those required subroutines that have not yet been loaded are loaded into the region/partition directly before subprogram initialization. Thus, there is only one copy of the needed subroutines resident in each region/partition.

For the dynamic CALL and CANCEL functions, the COBOL library management facility is an implied required feature. (See "Dynamic Subprogram Linkage.")

Programs written and compiled for the IBM OS Full American National Standard COBOL Program Product Compilers are compatible with OS/VS COBOL without recompilation, whether or not they use the COBOL library management facility.

SYSTEM/370 INSTRUCTION GENERATION

System/370 instruction generation is automatic. These System/370 instructions replace object-time subroutines and instructions that previous compilers generate under System/360—including routines and instructions to handle decimal arithmetic scaling (where operands have a different number of decimal places) and rounding. System/370 support also gives much improved processing of variable-length fields.

Because System/370 does not require boundary alignment for COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items, no internal moves are generated for items that are not SYNCHRONIZED.

Performance Considerations: Space occupied by an OS/VS COBOL program is decreased, particularly when calls to object-time subroutines are no longer necessary. Such calls are always generated in System/360 for variable-length moves and comparisons. If there is at least one variable-length alphanumeric move in the source program, System/370 support reduces the size of the object program by at least 484 bytes; if there is also at least one variable-length alphanumeric comparison, System/370 support reduces the size of the object program by at least an additional 498 bytes.

DYNAMIC STANDARD BLOCK SPECIFICATION

For queued sequential data sets, the RECFM subparameter of the DD statement may optionally be specified at object time, permitting the programmer to specify the standard block option (for data sets with recording mode F) or the track overflow option for the data set. (The track overflow option is equivalent to writing an APPLY RECORD-OVERFLOW clause in the source program.) Use of the standard block option (particularly for direct-access devices having the Rotational Positional Sensing feature) results in significant I/O performance improvement.

Fixed-block single volume data sets as created by COBOL are standard (except possibly when extended using the DISP=MOD parameter of the DD statement). Multivolume data sets as created by COBOL are standard if the volume switching occurs through automatic end-of-volume procedures. If, however, the programmer issues a CLOSE REEL/UNIT statement, then the number of logical records in the volume must be an integral multiple of _n_, where a BLOCK CONTAINS _n_ RECORDS clause (or an equivalent BLOCK CONTAINS—CHARACTERS clause) has been specified in the source program (that is, no truncated blocks exist). The standard block option and the track overflow option are mutually exclusive.

PRODUCTIVE COMPILE-TIME PERFORMANCE

SPEEDY SORTED CROSS-REFERENCE

If an alphabetized cross-reference list is requested, the OS/VS COBOL Compiler produces a cross-reference dictionary in which data-names, file-names, and procedure-names are sorted alphanumerically into two groups. One group consists of data-names and file-names; the second consists of procedure-names. Each is preceded by an appropriate subheading.

This option is requested at compile-time via the PARM field of the EXEC job control statement.

Note: A more comprehensive reformatted listing of the entire COBOL source program can be obtained by invoking the Lister facility. The listing thus produced contains embedded cross-reference information and simplifies tracing program logic. (See the preceding description of the Lister facility.)

OS/VS COBOL SUBROUTINE LIBRARY

The OS/VS COBOL Subroutine Library is a partitioned data set residing on a direct-access device, containing the COBOL object-time library subroutines in load module form. The OS/VS COBOL Subroutine Library is designed for use with object modules produced by the OS/VS COBOL Compiler. The OS/VS COBOL Subroutine Library is available as a separate Program Product. When more than one copy of the Subroutine Library is needed, this Program Product must be ordered.

COBOL library subroutines perform execution-time operations requiring either repetitive or extensive generated code. It is inefficient to place such code inline in the object module each time it is needed. Instead, library subroutines are used to reduce the size of the object module. Any library subroutines required to execute the problem program are either combined with the object module at link-edit time or are dynamically fetched during program execution.

To save even more virtual storage, the OS/VS COBOL library management facility allows a single copy of such COBOL object-time subroutines to be shared by problem programs in different partitions or regions. This is controlled by the user by placement of modules through a compile-time option. See the section describing the COBOL Library Management Facility.

There are several major categories into which the object-time subroutine library can be classified:

- Input/Output (excluding VSAM)
- Conversion
- Arithmetic verbs
- Other verbs
- Sort/Merge interfaces
- Checkpoint/Restart
- Segmentation feature
- Communications
- Debugging
- VSAM
- 3886 processing

The OS/VS COBOL Subroutine Library contains all subroutines needed to support the new features of the OS/VS COBOL Compiler.

COMPATIBILITY

DATA SET COMPATIBILITY

The OS/VS COBOL Release 2 Compiler provides support for indexed, sequential, and relative file capabilities through VSAM Release 2 as well as OS Full American National Standard indexes and relative file capabilities.

The VSAM compatibility allows the user to perform all equivalent ISAM-like functions on VSAM data sets using existing COBOL ISAM programs. When ISAM data sets are converted to VSAM format, the user can continue present operation with only minor JCL changes. The resulting VSAM data set usually requires less frequent reorganization than the ISAM data set, because VSAM uses free space within the data set for updating.

Data set compatibility (except for previously described VSAM data sets) exists between OS/VS COBOL and all versions of IBM OS Full American National Standard COBOL.

DOS/VS and OS/VS VSAM compatibilities are explained in DOS/VS Access Method Services. Common COBOL VSAM compatibilities do not affect these compatibilities.

PROGRAMMING COMPATIBILITY

For the most part, a single source program can utilize both 1974 Standard language elements and 1968 Standard language elements. When 1968 Standard language elements are used, the OS/VS COBOL Compiler gives results equivalent to those given by the IBM OS Full American National Standard COBOL compilers. For a few language elements, the two standards conflict; for these items, a compiler option is provided so that the programmer can instruct the compiler which language interpretation to use.

OS/VS COBOL incorporates all language elements from IBM OS Full American National Standard COBOL with two exceptions: The MESSAGE COUNT clause replaces the QUEUE DEPTH clause, and the ACCEPT MESSAGE COUNT statement replaces the IF MESSAGE statement.

Most programs written for previous versions of the IBM OS Full American National Standard COBOL compiler can be compiled on the OS/VS COBOL Compiler without modification, provided that new OS/VS COBOL reserved words have not been specified as user-defined names, the QUEUE DEPTH clause is not used, and the IF MESSAGE statement is not used. (A complete list of OS/VS COBOL reserved words is included in IBM VS COBOL for OS/VS.)

OBJECT PROGRAM COMPATIBILITY

Programs written and compiled for the IBM OS Full American National Standard COBOL Program Product Compilers are compatible with OS/VS COBOL without recompilation, whether or not they use the COBOL library management facility.

CMS COMPATIBILITY

Under the CMS component of VM/370, the OS/VS COBOL Compiler can accept and compile any COBOL source program that it can accept and compile under OS/VS1 and OS/VS2. The object code generated under CMS can be executed under control of OS/VS1 and OS/VS2. With restrictions listed in the following paragraphs, the object code can also be executed under CMS. The compiler is not aware of the CMS environment, and does not flag or identify the restricted usage for any operating system.

Compile-Time CMS Restrictions

The "(nn)" subparameter of the FLOW option must be specified; it is not optional. The "*" and "dsname" subparameters of the PRINT option, and the "dsname" subparameter of the LIB option must not be specified.

Execution-Time CMS Restrictions

- Indexed files (BISAM and QISAM) are not supported. Various clauses and statements associated with these access methods are therefore invalid: RECORD KEY, TRACK-AREA, START, APPLY REORG-CRITERIA, and APPLY CORE-INDEX.
- There is no Checkpoint/Restart feature. Therefore, the RERUN clause is not supported.
- The positioning options of the OPEN (EXTEND) and CLOSE statements are ignored.
- There is no multivolume data set support. Therefore, the CLOSE statement with the REEL or UNIT option is invalid.
- There is no TCAM support; however, the BSAM test facility will function for single level queues (i.e., not for queue structures).
- There is no Sort/Merge feature. Therefore, the SORT verb is not supported.
- None of the user label handling functions are supported. Therefore, the label handling format of USE is invalid. The data-name option of the LABEL RECORDS clause is invalid.
- ASCII-encoded tape files are not supported.
- Spanned recording mode is not available under QSAM, BDAM, and BSAM. This means that the S-mode default (block size smaller than record size) cannot be specified, and that the RECORDING MODE IS S clause cannot be specified.
- Neither the LISTING nor the SYSUT5 data set can be used under other systems.
- No support for the 3886 Optical Character Reader is provided.
- Creating direct files is restricted as follows:
  - For U or V recording modes, access must be sequential.
  - For ACCESS IS SEQUENTIAL, track identifier must not be modified.
  - The XTENT option of the FILEDEF command must be specified to indicate the number of logical records to be written.
- CALL...ON OVERFLOW statement is not available.

- No status key information on duplicate record using VSAM with alternate indexes.

- The GIVING option of the USE statement in the error declarative section is not supported for VSAM data sets.

- The AIXBLD execution-time option is not supported. Therefore, the dynamic building of alternate indexes and the dynamic completion of VSAM relative record data sets (RRDS) record information is not supported.

SYSTEM REQUIREMENTS

Operating System Needed	OS/VS1[1] OS/VS2 (with or without TSO)[1] MVS/XA (24-bit mode only) CMS component of VM/370[1]
Compile-time Machine Requirements	Any System/370 model that supports OS/VS or CMS[2] **Disk Work File** SYSUTI (required) **Tape or Disk Work File** SYSUT2—SYSUT4 (required) SYSUT5—if Symbolic Debug is used SYSUT6—if FIPS flagger is used
Object-time Machine Requirements	Any System/370 model that supports OS/VS or CMS[2] One Work File (tape or disk) when Symbolic Debug is used Input/output devices used by the object program
Virtual Storage	128K bytes for Release 2 of the Compiler

Notes

[1] Release 2 of VSAM is required for execution of COBOL object programs that use VSAM alternate index, or relative record. Use of VSAM file status feedback when creating duplicate alternate record keys or use of QSAM file EXTEND requires at least OS/VS1 Release 6.0, OS/VS2 Release 3.7 (MVS) plus SU8, or OS/VS2 Release 1.7 (SVS) Extended. TCAM Mod Level 5 is required if the Communications Feature is used.

[2] For CMS restrictions, see the CMS Compatibility section.

OS/VS COBOL LANGUAGE BASE

OS/VS COBOL is designed according to the specifications of the highest levels of the following 1974 Standard modules:

NUCLEUS—provides improved internal processing capabilities. Extended data manipulation statements, enhanced arithmetic capabilities, user-specified collating sequences, and eased data grouping rules are all provided.

TABLE HANDLING—a convenient method for searching a table is provided which allows the definition of tables and making reference to them through subscripts, mixed indexes, and literal subscripts.

SEQUENTIAL I-O—implements the VSAM ESDS processing, allows sequential files to be extended, and provides added page placement capabilities for physical sequential files destined for printed output.

RELATIVE I-O—allows the user to specify relative record file organization—the records in such a file are stored and retrieved in the order of their relative record numbers. Storage and retrieval can be sequential or random. Relative I-O is implemented through the VSAM RRDS capabilities.

INDEXED I-O—gives added indexed file processing capabilities through VSAM KSDS processing. Records are stored according to some prime record key; they can be retrieved through the prime record key or through alternate record keys. Storage and retrieval can be sequential or random.

SORT-MERGE—gives the capability of ordering the records in one or more files (sorting) and of combining two or more identically ordered files (merging). The sort or merge can be upon either the EBCDIC or ASCII collating sequence, or upon a user-specified collating sequence.

SEGMENTATION—allows the user to specify object program overlay requirements.

LIBRARY—allows the user to replace all occurrences of a given library text with alternate text during compilation. Multiple COBOL source libraries can be specified.

DEBUG—provides the capability of monitoring object program execution through Declarative procedures, special debugging lines (executed only when in debugging mode), and a special register DEBUG-ITEM which gives specific information about execution status.

INTER-PROGRAM COMMUNICATION—allows a COBOL program to communicate with one or more other programs through transfers of control and access to common data items.

COMMUNICATION—provides the ability to communicate through a Message Control Program (MCP) with local or remote communication devices, and to access, process, and create partial and complete messages.

OS/VS COBOL also supports the highest levels of all eight modules of 1968 Standard COBOL. (Language in the 1968 Standard that differs from 1974 Standard COBOL is considered an extension to the 1974 Standard.) These eight modules are:

NUCLEUS—defines the permissible character set and the basic elements of the language contained in each of the four COBOL divisions. Identification Division, Environment Division, Data Division, and Procedure Division.

TABLE HANDLING—allows the definition of tables and making reference to them through subscripts and indexes. A convenient method for searching a table is provided.

SEQUENTIAL ACCESS—allows the records of a file to be read or written in a serial manner. The order of reference is implicitly determined by the position of the logical record in the file.

RANDOM ACCESS—allows the records of a file to be read or written in a manner specified by the programmer. Programmmer-specified keys control successive references to the file.

SORT—provides the capability of sorting files in ascending and/or descending order. This feature also includes procedures for handling such files both before and after they have been sorted.

REPORT WRITER—allows the programmer to describe the format of a report in the Data Division, thereby minimizing the amount of Procedure Division coding necessary.

SEGMENTATION—allows large problem programs to be split into segments to be designated as permanent or overlayable storage. This assures more efficient use of storage at object time.

LIBRARY—supports the retrieval and updating of prewritten source program entries from a user's library, for inclusion in a COBOL program at compile time. The effect of the compilation of library text is as though the text were actually part of the source program.

In addition, OS/VS COBOL includes IBM extensions that provide programmer convenience and additional processing capabilities. The major IBM extensions are:

PASSWORD CLAUSE—provides file security for physical sequential and VSAM files.

TRANSFORM STATEMENT—provides easy translation capabilities from one collating sequence to another.

ENTRY STATEMENT—gives the user the ability to specify alternate entry points in a called program.

RELATED COBOL DEVELOPMENT AIDS

In addition to the program development features included within the OS/VS COBOL Compiler and Library itself, there are two related IBM Program Products, available under TSO, that greatly facilitate program development. Both reduce program turnaround time and increase programmer productivity. These Program Products are: the TSO COBOL Prompter and COBOL Interactive Debug.

Both are described in IBM OS COBOL Interactive Debug and (TSO) COBOL Prompter, General Information.

The following discussion describes how these Program Products can be used with OS/VS COBOL to make effective use of all three.

TSO COBOL PROMPTER

The TSO COBOL Prompter makes possible conversational OS/VS COBOL compilations from a terminal. The Prompter functions exactly as its name implies: if the terminal user has omitted necessary compilation information, or has entered such information incorrectly, the Prompter asks for the correct information. This saves time, effort, and expense, because, at one terminal session, the OS/VS COBOL program can achieve results that might take several batch processing runs.

PROMPTER FUNCTIONS

- Accepts terminal input—including optional compilation parameters and the source data set name
- Analyzes the input and prompts the terminal user
- Dynamically allocates data sets—both those required by the Compiler, and any that are optional
- Builds option and ddname lists for the Compiler
- Invokes the Compiler, supplying any user-omitted items

During a terminal session, the user will receive both informational and prompting messages.

Prompting messages ask the user to correct erroneous compilation information. When the information is corrected, the terminal session resumes.

The user can also request assistance from the Prompter in using the Prompter itself.

COBOL INTERACTIVE DEBUG

COBOL Interactive Debug greatly simplifies the debugging of OS/VS COBOL object programs by providing facilities which make any error readily apparent and easily correctable. COBOL Interactive Debug is available under the TSO option of OS/VS2 and under the CMS component of VM/370.

During one session at the terminal, without changing source code and without recompilation, the user can dynamically trace the path of program flow, temporarily alter the logic flow of the program, execute and reexecute portions (or all) of the object program using different data values, and inspect the contents of data items at any selected point in the program. Many other features are also available. The result is that one terminal session can take the place of many batch debugging runs; this saves both machine time and programmer time, and speeds up program development.

INTERACTIVE DEBUG FUNCTIONS

The Interactive Debug user can dynamically monitor the execution of his COBOL program from a terminal. As execution of the object program proceeds, the user, through the specification of Interactive Debug subcommands, can dynamically:

- Establish breakpoints at which program execution is to be unconditionally suspended and control returned, and remove breakpoints already established.

- Establish conditional breakpoints which are effective only when certain conditions are met. Such conditions might be, for example, the debug monitor detecting a change in value of a data field, or, a valid relational test involving two data fields or a data field and a literal.

- Specify that at a breakpoint certain actions will automatically take place (for example, changing the value of a data item), after which execution will automatically be resumed (that is, patch the code).

- List the active breakpoints.

- Resume execution at a specified source procedure-name.

- Trace the execution of the object program by requesting a display of the names or line numbers of all source procedures through which control has passed.

- Display selected COBOL source statements.

- List the status of files in the program.

- Display/compare/modify the contents of data items in the COBOL program.

- Obtain a system dump of the problem program region.

- Request information about the function, syntax, or operands of the Interactive Debug subcommands.

INDEX

IBM OS/VS COBOL Compiler and
Library General Information
GC28-6470-2

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ____________________

Previous TNL ____________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

GC28-6470-2

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape

Please do not staple

Fold and tape

IBM OS/VS COBOL Compiler and Library General Information (File No. S370-26) Printed in U.S.A. GC28-6470-2

IBM OS/VS COBOL Compiler and
Library General Information
GC28-6470-2

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ________________

Previous TNL ________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape

Please do not staple

Fold and tape

IBM®

GC28-6470-2

IBM OS/VS COBOL Compiler and Library General Information (File No. S370-20) Printed in U.S.A. GC28-6470-2