LY28-6422-2

Program Product

VS BASIC Program Logic

Program Number 5748-XX1

IBM

Third Edition (December 1976)

This edition applies to Release 3 of VS BASIC, program number 5748-XX1, and to any subsequent releases unless otherwise indicated in new editions or technical newsletters. Release 3 will run under the same operating-system environments that support the current version and modification level.

The changes for this edition are summarized under "Summary of Amendments" following the list of illustrations. Technical changes made are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the pages affected. Editorial changes that have no technical significance are not noted.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

PREFACE

This publication describes the internal logic of the IBM System/370 VS BASIC Processor. It is primarily intended for customer engineers and other technical personnel involved in program maintenance. Program logic is not necessary for the use and operation of the System/370 VS BASIC processor; therefore, distribution of this publication is limited to licensees who have the aforementioned requirement.

This publication consists of the following sections:

Section 1: Introduction

The "Introduction" presents a broad overview of the VS BASIC processor, its operation and its major components.

Section 2: Method of Operation

The "Method of Operation" section describes in HIPO format the functions that the VS BASIC processor performs. HIPO is a pictorial method for describing the function of a program. It shows the input, process, and output required to perform a particular function.

Section 3: Program Organization

The "Program Organization" section illustrates the flow of control from component to component. It contains a tabularized guide to the hierachical structure of the processor.

Section 4: Directory

The "Directory" serves as a guide to the PLM. It lists, in tabular form, the names of the components of the processor and where they are referred to in the "Method of Operation" section.

Section 5: Data Areas

The "Data Areas" section outlines in tabular and pictorial form the various data areas used by the processor for communication between components. It lists the displacements of the areas and shows how they look in storage.

Section 6: Diagnostic Aids

The "Diagnostic Aids" section contains information that is useful in isolating a problem and examining the contents of storage.

Section 7: Appendixes

The "Appendixes" section contains examples of the executable code produced by the VS BASIC compiler and the text elements produced by the scanning routines of the debug processor.

Reference Publications

It is assumed that the reader has a thorough knowledge of the VS BASIC language as described in:

VS BASIC Language
Order No. GC28-8303

And the system under which VS BASIC will be running as described in one of the following publications:

VS BASIC
TSO Terminal User's Guide
Order No. SC28-8304

VS BASIC
CMS Terminal User's Guide
Order No. SC28-8306

VS BASIC for VSPC:
Terminal User's Guide
Order No. SH20-9060

VS BASIC
Batch Programmer's Guide
for OS/VS, DOS/VS
Order No. SC28-8308

Although not required, the following publication provides related information:

VS BASIC
Installation Reference Material
Order No. SC28-8309

If more detailed information is required, the reader should refer to the comments and coding in the System/370 VS BASIC processor listings.

CONTENTS

ILLUSTRATIONS

FIGURES

TABLES

DIAGRAMS

NUMBER 2

The "Introduction" section includes a brief discussion of what occurs when the TEST option is not in effect, and the error handling capabilities of the new ON statement and I/O error clauses in the VS BASIC run-time library.

The "Method of Operation" section reflects changes to the Compiler Processing diagram and discussion to show the new OPTION statement. The Compiler--Statement Processing diagram and discussion includes new statements INPUT FROM, PRINT TO and ON. The Run-time Processing diagram and discussion includes changes to the modules ICDKORGE, ICDKxSUB (x is D, G, or S) and ICDKERR. The Run-time Terminal I/O diagram and discussion incorporates changes to the modules ICDKINPT, ICDKPRNT, to reflect the new statements INPUT FROM and PRINT TO. Also, new data area BUFFAHED is incorporated to show the new buffered-ahead terminal input facility. The Run-time Record I/O diagram and discussion incorporates the new relative-record file capability and the implicit open for terminal files.

The "Program Organization" section has been amended to reflect the new entry points of the processor modules and flow of control to and from each entry point for the new VS BASIC statements, error handling capabilities and the new intrinsic function (CHR).

The "Data Areas" section has been amended to include addition of new labels within the following data areas: PRG, VARCON, VFILTAB, ICDBIFTB and to the data areas directory.

The "Diagnostic Aids" section incorporates the new error message identifiers for the library and executors. Also, the VS BASIC OS/VS System Abnormal termination code for object code incompatibility is given.

Appendix A has been amended to include examples of the object code produced by the VS BASIC processor for the new VS BASIC statements, INPUT FROM, PRINT TO, ON, and changes to the code for OPEN, CLOSE etc. for the new error conditions.

NUMBER 1

This edition includes VSPC as one of the systems that support the VS BASIC processor. The "Introduction" section includes a description of the VS BASIC executor. This description references the one for the TSO executor, as do the other executor descriptions. The other sections of the book include updates for VSPC.

The "Method of Operation" section has been replaced in its entirety, and the diagrams have been condensed for usability. (The "Directory" shows new references for the diagrams.)

The "Data Areas" section has been amended to include the ESPACE control block. ESPACE is used by the run-time library I/O routines. Offsets in PRG have been changed to reflect the code.

The "Diagnostic Aids" section includes requirements for writing an APAR.

"Appendix A" has been amended so that a description of the object code shows consistently the register-naming conventions used by the VS BASIC processor.

Miscellaneous minor technical corrections have also been made throughout the book.

GENERAL DESCRIPTION

The VS BASIC Processor is designed to operate in either an interactive or a batch virtual environment. The processor can be logically divided into four parts: an executor, a compiler, a library, and a debug processor.

The executor serves as an interface between the system under which VS BASIC is running, and the other three parts of the processor. It insulates the processor from the system and permits it to operate without any dependence on the host system. The executor intercepts and relays any processor requests for system services.

The compiler is a fast, one-pass language translator that accepts source programs written in the VS BASIC language and translates them into object code that is suitable for loading and executing under a VS BASIC executor on a System/370 machine. Optionally, the compiler will accept source code in long or short precision, permit the compilation to proceed into execution, store the object code produced, or produce object code that has been tailored to meet the needs of the debug processor. Since the compiler is reentrant, it can be installed in the link pack area making it available to a number of users simultaneously.

The library contains run-time routines that assist in the execution of VS BASIC programs. In addition, it also contains routines that execute intrinsic library functions.

The debug processor permits the user to set breakpoints in his program as it is executing, display the contents of his program variables, and to trace the flow of control through the program. It is available only under the TSO and CMS interactive systems.

Under TSO and CMS a renumbering facility is available for renumbering VS BASIC source programs. The VSPC service program performs importing functions to include VS BASIC programs and data in the VSPC data base. Under CMS, a conversion facility is available to convert CALL-OS BASIC data files to a VS BASIC format.

PROGRAMMING SYSTEM ENVIRONMENTS

The VS BASIC Processor will operate in a variety of interactive and batch environments. They are:

INTERACTIVE

- TSO under an OS/VS2 system that optionally supports VSAM.
- CMS under a VM/370 system that optionally supports VSAM.
- VSPC under OS/VS2, OS/VS1, or DOS/VS system that supports VSAM, VTAM.

BATCH

- An OS/VS1 system that optionally supports VSAM.
- An OS/VS2 system that optionally supports VSAM.
- A DOS/VS system that optionally supports VSAM.
- CMS under a VM/370 system that optionally supports VSAM.

EQUIPMENT CONFIGURATION

VS BASIC in an interactive or a batch environment, runs on a System/370 Model 135 or larger. In a batch environment, VS BASIC runs on a System/370 Model 115 or larger.

STORAGE REQUIREMENTS

The VS BASIC Processor requires 120K bytes of virtual storage under VSPC, 128K under OS/VS, 256K under DOS/VS, and a virtual machine size of 300K under VM/370 (CMS).

VS BASIC PROCESSOR OVERVIEW

The VS BASIC Processor is made up of four components: the executor, the compiler, the run-time library, and the debug processor. The executor serves as an interface between the system under which VS BASIC is operating and the compiler (during compilation) or the library and debug processor (during execution). Unlike the other components, the executor is tailored to the system it operates in. Therefore, five versions of the executor are available: one for VSPC, one for TSO, one for CMS, one for OS/VS1 and OS/VS2 batch, and one for DOS/VS batch. The compiler, identical for all systems, takes source programs written in the VS BASIC language and translates them into executable object programs that are suitable for loading and executing by a VS BASIC executor. The run-time library, like the compiler, identical for all systems, assists in the execution of object programs. The library performs services such as evaluating VS BASIC intrinsic functions and completing the execution of VS BASIC statements either directly or through requests to the executor. The debug processor can be optionally invoked in the interactive environments of TSO and CMS. It permits the VS BASIC programmer to analyse and to monitor the execution of his source program and to examine its processing as it is being performed.

VS BASIC EXECUTOR

TSO VS BASIC Executor

The operation of the TSO executor can be divided into three separate phases. The first (initialization) establishes and initializes data areas needed by the compiler and run-time library. The second (service call processing) handles requestsfor system services, including stream and record input/output. The third (special purpose processing) includes exit and termination routines.

EXECUTOR INITIALIZATION: The VS BASIC executor receives control from the TSO Terminal Monitor Program or from the TSO EDIT Command Processor. The executor is passed, as a parameter, a pointer to a buffer that contains the user's RUN command. When control is received, a workspace is prepared by issuing a GETMAIN. This workspace contains a register save area, a data storage area, and a buffer for formatted terminal output. It is accessed through a DSECT that is based on register 13; therefore, any routines that are subsequently called will be able to use the save area when they receive control from the executor.

The executor then loads the TSO DAIR and PUTLINE/GETLINE/PUTGET service routines, assuring that they will be available in the user's TSO region. The executor links to the TSO parse routine and passes it the pointer to the command buffer that it received originally. The PARSE routine checks the command for syntax and returns a Parameter Description List (PDL) to the executor.

The PDL is then analysed. The first item to be determined is the name of the VS BASIC program to be processed. If the program was just created under the EDIT command and is still in storage, the PDL contains the INLIST keyword, which, in turn, contains the address of the parameter list pointing to the name. If the INLIST keyword is not present the name of the program is in the command buffer. At this point, a header message is prepared using the program name and date and time information that was obtained from the TIME and STIMER macros. This header message is then printed at the terminal by the TOUTPUT subroutine of the executor. SPIE, STAX, and STAE macros are issued to specify abnormal and special exit routine. (These routines are discussed further in the section "Special Purpose Subroutines.")

The PDL is examined again. If the SOURCE option is found, or if the INLIST keyword had been found previously, the compiler is loaded into storage and a GETMAIN is issued for the user area. The GETMAIN is for the amount of storage specified in the SIZE option or for all the storage that is available up to a maximum of 510K, if the SIZE option has not been specified. The source statements are then moved into the user area one record at a time.

As each statement is read, it is transformed into the following format:

Statement Length + 2	Source Statement (without trailing blanks)	X'15'
1 byte	n bytes	1 byte

If the source program is already in storage, it is not moved but converted inplace. The source statements are then placed into the user area immediately after the user terminal buffer. When all source statements have been read a X'01' is placed at the end of the last record. The number

of source lines and the number of bytes in the program are placed in the User Terminal Table (UTT) in the user area. The entire program is then moved to the end of the user area. The size of the available user area is reduced by the amount of space occupied by the source code.

If the OBJECT option has been specified, and the GO and NOSTORE options have also been specified, a data set name is created by DAIR with the name entered by the user. This data set is then opened. The data set is read into the beginning of the user area and the unused portion of the user area is released by a FREEMAIN macro. Control is then passed to the part of the executor that will initiate execution of the object program. In cases where the option SOURCE has been specified or the option OBJECT has not been specified, and the executor is unable to find a corresponding source program, an object program, if found, will be executed instead.

EXECUTOR SERVICE CALL PROCESSING: During compilation and execution, the VS BASIC processor obtains system services through the executor by means of its own unique set of service call macros. When issued by the compiler or library, the executor will interpret the call, supply the requested service, and pass control to the appropriate processor or system routine, the service call (SERV) macros are:

- SVC0 - Normal termination of execution
- SVC1 - Terminal output
- SVC2 - Terminal input
- SVC3 - Stream file output
- SVC4 - Stream file input
- SVC5 - Release storage
- SVC6 - Acquire storage
- SVC7 - Return from arithmetic interrupts (compilation)
- SVC8 - Return from arithmetic interrupts (execution)
- SVC9 - Encode file name
- SVC10 - Stream file output and close
- SVC11 - End of compilation
- SVC12 - Time/Date function
- SVC13 - PAUSE statement processing
- SVC14 - Terminal input
- SVC16 - CHAIN statement processing
- SVC18 - Task CPU time
- SVC21 - Stream file open
- SVC22 - Stream file close
- SVC23 - Abnormal termination of execution
- SVC24 - Record file input/output
- SVC25 - Record file open
- SVC26 - Record file close

SPECIAL PURPOSE SUBROUTINES: Special purpose processing consists of various return routines that are called for either normal or abnormal returns or end of processing. The SVCRET routine returns control to the compiler after a request has been made for system services. The CLEANUP routine releases all resources before a final return is made to TSO following the completion of processing. The three exit routines, STAEEXIT, STAXEXIT, and SPIEEXIT, process unusual or error conditions: too little disk space, for the output data set, attention interrupts, and program checks respectively.

OS/VS1 and OS/VS2 Batch Executor

The operation of the OS/VS executors is similar to that of the TSO executor. The major difference between them is the presence of a CONTROL data set on punched cards, that contains RUN commands as records and the need for JCL to support all data set references.

The OS/VS batch executor receives control from the operating system, and, after a GETMAIN is issued for the executor's workspace, the SYSPRINT and the CONTROL data sets are opened. The RUN commands in the CONTROL data set are processed one at a time. The batch executor has its own code (CSECT CNTRLREC) to scan the RUN commands for a valid program name, valid file names, and to analyse the requested options for both compilation and execution. After scanning the RUN command, the executor opens all the requested files (source/object program, input data, output object program). All files with appropriate JCL must be available to the executor in either the job stream or as cataloged data sets. Depending upon the options specified, either the compiler or the run-time library is loaded and the remainder of the executor's initialization is completed.

The service call processing of the batch executor is identical to that of the TSO executor except that all requests for terminal input and output (SVC1 and SVC2) are translated into reads from the input file (as designated on the RUN command) and writes to SYSPRINT.

The special purpose subroutines are also identical to the TSO routines with the exception of the terminal oriented functions (that is, there is no STAX handling code and PUT/GET is not loaded or deleted).

CMS VS BASIC Executor

Functionally, the VS BASIC executor under CMS is similar to the executor under TSO. The real differences occur in the way in which CMS processes requests from the executor; however, even these are minimized since CMS recognizes the OS GETMAIN, FREEMAIN, and BSAM macros.

The VS BASIC executor receives control from the CMS command processor, and is passed a parameter list containing double word entries for the filename, left parenthesis, and the options. The SIZE option cannot be specified under CMS as all available storage in the user's virtual machine up to a maximum of 510K is obtained immediately and the components of the user area are stored contiguously.

VSPC Executor

The VSPC executor performs initialization, service call, and special purpose functions similar to the TSO executor. However, the VSPC subsystem issues all system service requests and provides access to virtual storage for VS BASIC.

INITIALIZATION: When the executor receives control, VSPC passes it a pointer to the perterm communication block (PTC), which contains the information required to process a user's program. The PTC request code field tells the executor whether to compile or execute a program or whether an asynchronous interrupt occurred during program processing. The request codes are IBEGIN, IBGNX, and IASYNC, respectively. The information in the PTC includes initialization values for the communication area, workspace header, and UTT. For compilation, the executor moves the source program to the end of the user's workspace and calls the compiler.

SERVICE CALL PROCESSING: The executor issues the VSPC ASUSRQ macro to obtain system services. The request code name with the macro identifies the specific request. The request codes used by the VSPC executor are grouped below according to the type of service request:

- VSPC Library Access -- DCLOSE, DFNCDE, DOPEN, DREAD, DRESET, DWRITE
- VSAM Library Access -- VCLOSE, VDEL, VOPEN, VREAD, VRESET, VREWRT, VRREAD, VROPEN, VTCLS, VWRITE
- Terminal Services -- TDELAY, TWRITE, TWRTRD
- Workspace Management -- WCHAIN, WDUMP, WENDC, WENDR, WEXIT, WFREEM, WGETM
- Host System Service -- HACCT, HSYSLG, HTIME

Since the VSPC subsystem can relocate the workspace before returning to the VS BASIC executor, the executor checks the workspace address in the PTC. If relocation occurred, addresses and relocatable registers are adjusted as necessary before returning to the processor.

DOS/VS Batch Executor

The DOS/VS executor is similar to the OS/VS executor. There is no separate CONTROL file, and, in addition, both the source or object programs as well as the input data file must be made available to the executor through SYSIPT.

The job stream on SYSIPT consists of RUN commands similar to the OS/VS RUN commands in the CONTROL data set; however, an * is used in place of the program name. The input file name is followed by the source or object program, which, in turn, is followed by the input data, if any. The end of the source program is delimited by an identifier card that contains a blank in columns one and three and a slash in column two.

The DOS/VS executor uses DTFDI and DIMOD for the system files SYSLST, SYSPCH, and SYSIPT. DTFCP and CPMOD are used for all stream input/output files.

VS BASIC COMPILER

The VS BASIC compiler is reentrant and one-pass. Reentrant means that one copy of the compiler can be shared by many user's programs in different stages of compilation. This results in an important saving of time, since, during a page change, the compiler code need not be written out; a new copy of the compiler can be read in when needed. This is accomplished by not allowing the compiler to modify itself in any way and by storing all variables and tables used during compilation separately in the user area. One-pass means that the compiler makes only one scan through the source statements to produce executable object code rather than going over the source statements many times before the object code is produced.

The VS BASIC compiler is organized into three processing phases: compiler initialization, statement processing, and run-time initialization.

Compiler Initialization

Compiler initialization is the first and shortest phase of compilation. This phase reallocates the user area into a form required by the compiler and initializes values in order to start the compilation process. A single module, ICDJCMPA, handles the operation of this phase. Figure 1 illustrates the user area before and after the restructuring has taken place.

The executor creates a user area that contains, initially, a communications region and a terminal buffer (for TSO and CMS) or an output buffer (for OS and DOS batch) at the top (low storage addresses) and the VS BASIC source program at the bottom (high storage addresses). ICDJCMPA takes the intervening space and allocates it into fixed blocks.

If storage is exceeded, the compilation terminates. The blocks are placed in the user area so that the blocks that are required for both compilation and execution are near the top, and the blocks that are required only for compilation are near the bottom of the user area. The blocks near the bottom can be easily overlaid by blocks that are required only for execution. These blocks are allocated by the run-time portion of the compiler. The unused portion of the object area, the overflow area, the work area, the compiler tables area, and the source code are replaced or released at the end of the compilation. Array storage is allocated at the beginning of run-time execution. Since VS BASIC does not permit data initialization of arrays, all that is needed during compilation is to determine the amount of array storage required and the offset of a particular array in the eventual array area.

ICDJCMPA initializes the blocks in the user area as follows:

- Initializes data entries in the communications region to enable interactions between the compiler and executor.
- Initializes the values that are required by the compiler. The area used for program constants and variables, will be initialized to zero or blank, as required by the data type.
- Initializes tables for the start of compilation.

Statement Processing

The bulk of the compiler's processing time is spent in the statement processing phase

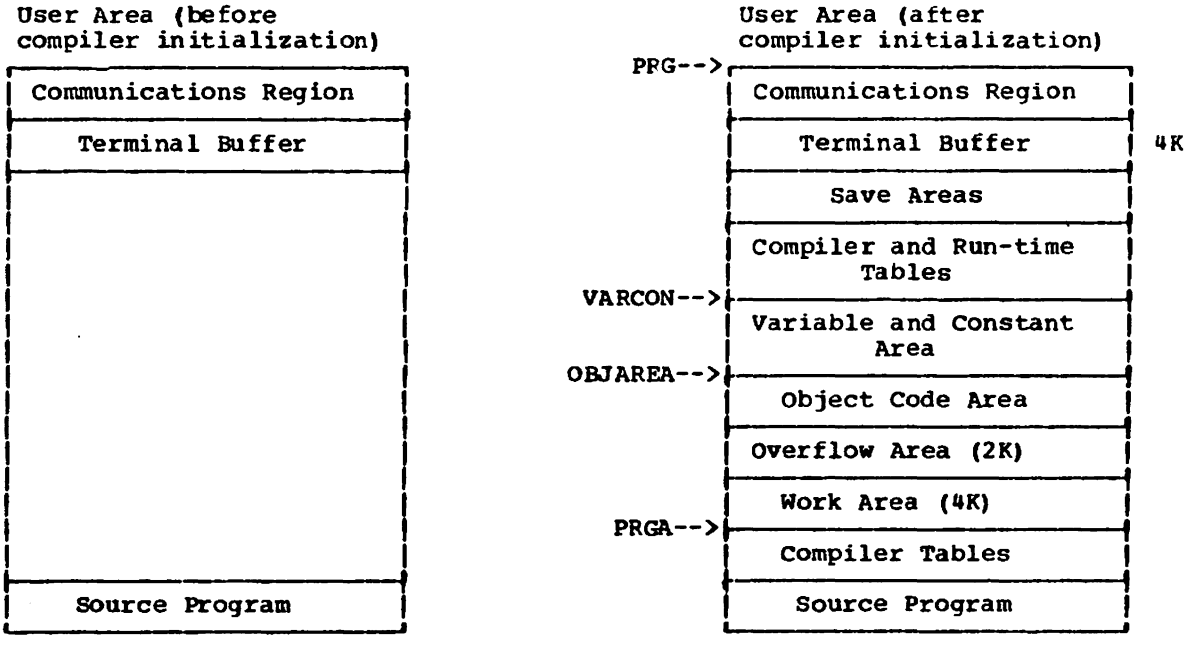


Figure 1. Structure of the User Area before and after Compiler Initialization

of operation. Statement processing consists of translating VS BASIC source statements into executable object code. The statement processor control routine, ICDJCTL, identifies each statement and passes control to the appropriate statement processing routine.

The compilation of any DATA, EXIT, FORM, or Image statements encountered in the program is deferred until the end of compilation, when control is transferred to the deferred compilation routine ICDJDEFR.

During their operation, the statement processing routines may call other service and utility routines that evaluate expressions (ICDJFMLA), scan source statements for variables (ICDJSCN), or perform conversion of numeric constants (ICDJCONF). The object code emitted by the statement processing routines is placed, sequentially, into the object code area as it is produced. The code is of two types. The first type contains all the code necessary to completely execute the source statement. The second type, however, does not; it contains a branch to a run-time library routine that will complete the execution of the statement.

When a statement processing routine has completed processing a source statement, control is transferred back to the control routine, which begins the compilation cycle over again. When it receives control, ICDJCTL first determines the line number of the statement to be processed and locates the address in the object code area for the resultant translated code. It keeps a table for the line numbers with displacements to the corresponding object code. ICDJCTL also removes blanks that have no special significance. After the blanks have been removed, the control routine determines the type of statement it is handling and the location of the processing routine that will translate it. ICDJCTL then transfers control to the processing routine. After the last statement has been translated, control is transferred to the run-time initialization phase of the compiler.

If the TEST option is specified under TSO or CMS, each compiler statement processing routine inserts additional code to transfer to the debug processor before executing the normal object code produced.

Compiler Run-Time Initialization

The run-time initialization phase of the compiler begins by restructuring the user area for execution, if no errors were detected during compilation. Here, the compiler routine ICDJRUNA together with the library routine ICDKORGE handle run-time initialization. If any errors were detected during compilation, ICDJRUNA transfers control to the executor to print out any diagnostic messages and, for most errors, to terminate processing. Otherwise, data areas in the PRG section of the user area that are used only during compilation are replaced by data areas that are used only during execution. ICDJRUNA passes control to the executor indicating the end of compilation. The executor releases the VS BASIC compiler (TSO only) and transfers control, after loading it, to the run-time library initialization routine (ICDKORGE) to begin execution of the object program.

VS BASIC Run-Time Library

After completion of the compiler run-time initialization, control is passed, by the executor, to the run-time library initialization routine, ICDKORGE. ICDKORGE restructures the remainder of the user area. The unused portion of the object code area, the overflow area (if not used), the work area, the compiler tables, and the source program are replaced by array storage and disk buffers. Figure 2 illustrates the restructuring of the user area before and after run-time initialization has taken place.

In addition, if the TEST option has been specified, ICDKORGE calls ICDBLDTB to build tables for use by the debug processor when execution is begun.

Execution of the object code begins at the first executable statement and continues sequentially until the end of the program is reached. If the TEST option is in effect, code at the beginning of each executable statement transfers control to the debug processor. If the TEST option is not in effect, code at the beginning of each executable statement establishes addressability for that statement, and monitors attention interrupt processing. Code for statements that were completely compiled is executed directly; code for statements that were not completely compiled contains branches to the appropriate library routine.

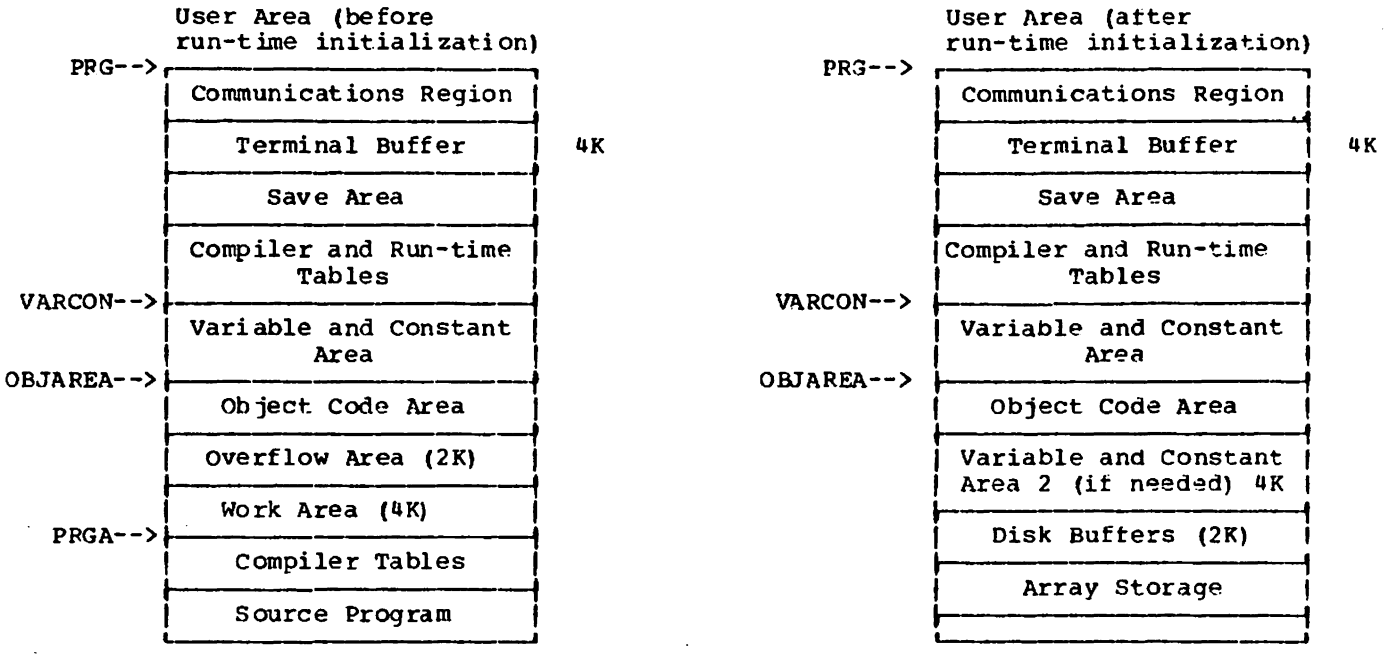


Figure 2. Structure of the User Area before and after Run-Time Initialization

The run-time library routines can be grouped into three types. The first type are routines that complete the execution of a VS BASIC statement. The object code for the statement contains a link to the run-time library, which returns to the object code after its processing is complete. The second type of routine evaluates a library function. The object code for the function reference contains a link to the library routine that evaluates the requested function. The third type of routine provides services for the function reference contains a link to the library routine that evaluates the requested function. The third type of routine provides services for the other types of routines.

The library also handles errors that occur during execution. Arithmetic interrupts are passed to the executor which returns control to the library. Execution errors are handled by the library routine ICDKERR. This routine normally prints out the corresponding messages and passes control to the executor to terminate execution.

However, the user, through the use of the ON statement or I/O error clauses, can specify a routine to which control is passed if certain errors should occur.

When the code for the STOP or END statement is reached under normal circumstances, control is transferred to the executor to terminate execution normally.

VS BASIC Debug Processor Operation (TSO and CMS only)

Whenever the TEST option has been specified, the compiler and the run-time library perform initialization and emit code in preparation for the operation of the debug processor. The compiler places code that will branch to the debug processor at the beginning of each translated executable statement. The routine ICDBLDTB builds tables of information that debug will refer to during its processing. When control is transferred to the object code and the branch to debug is present, debug begins executing. On the first branch, the user is requested to enter debug commands. As the commands are received, they are processed in two phases: scan and obey. The scan phase is controlled by a central dispatch routine, ICDSCAN, which obtains the command and calls ICDDSCAN to transfer control to the particular routine that will actually carry out the scanning. Once the command has been scanned, control is passed to the routine, ICDOBEY, that will cause the command to be executed. Linking the processing of these two routines together is a monitor routine, ICDONITR, which updates the communications region, processes some statements, and issues trace messages.

Renumbering Facility (RENUM Subcommand)

The renumbering facility interface routine ICDQRNME is invoked as an exit routine of IKJEBHRE (RENUM subcommand of EDIT) when it finds a valid renum command for a VS BASIC data set. The VS BASIC RENUMBER routine is passed the renumbering values, the in-storage data set, its size and attributes. To perform the renumber operation two passes of the in-storage data set are required. The first pass builds a number table that contains the existing line numbers and each corresponding new line number. The second pass allocates a new in-storage data set that will contain the renumbered source, then each line is copied to a buffer with the new line number replacing the old one. This buffer is passed to a scanning routine (ICDQRNMS) to check for statement number references which are replaced by the corresponding new line numbers from the number table.

ICDRNMS is passed an input buffer containing the source statement, a work area and an output buffer. Blanks are squeezed from the input buffer to allow scanning. Then the squeezed source statement, now in work buffer 1, is checked for statement number references and when found they are replaced by the new line numbers in the number table. The updated and squeezed source, now in work buffer 2, is then expanded into the output buffer by comparing the original record with the squeezed record to insert blanks where required.

The updated line then is placed in the new in-storage data set. On successful completion the in-storage data set size and location are updated to point to the new in-storage data set. On exit to IKJEBERE the number table and unused in-storage data set are freed. ICQRNME supports both fixed length and variable length records in the in-core data set. Standard line numbers are in columns 1-5 of the statement. For nonum data sets, the line number must start in column 1 and be any length up to a maximum of 9 digits. If the first record of a fixed format, nonum data set, does not have a valid line number on the left of the record, the program assumes the line number is 8 bytes long and is at the end of the record. This provides for conversion of TSO ITF BASIC data sets which are fixed length format and data sets created and numbered using IEBUPDTE (or a similar offline utility that renumbers fixed length records in the last 8 bytes of the record.)

CMS File Conversion Utility (ICDLUTIL)

The user invokes the conversion utility by issuing the name of the utility (ICDLUTIL) as a CMS command. The command includes the names of the files to be converted. CMS processes the command and builds a parameter list of the file names, which it passes to the conversion utility. A STATE macro is issued for each file on the list to determine whether the file exists. If the file exists, it is checked to determine if the file contains fixed-length records (F) with a logical record length of 80. All other types of files are invalid as input to the conversion utility.

A FILEDEF command is issued for each file and an attempt is made to open them. If the files can be successfully opened, corresponding output files are defined and opened. After this is done, the first record is read in. The length of the record is obtained from the first four bytes and the end-of-block (EOB) address is calculated. The address of the output buffer is obtained and an end-of-block address is calculated for it. The first four bytes of the output buffer are reserved for the record descriptor word (RDW).

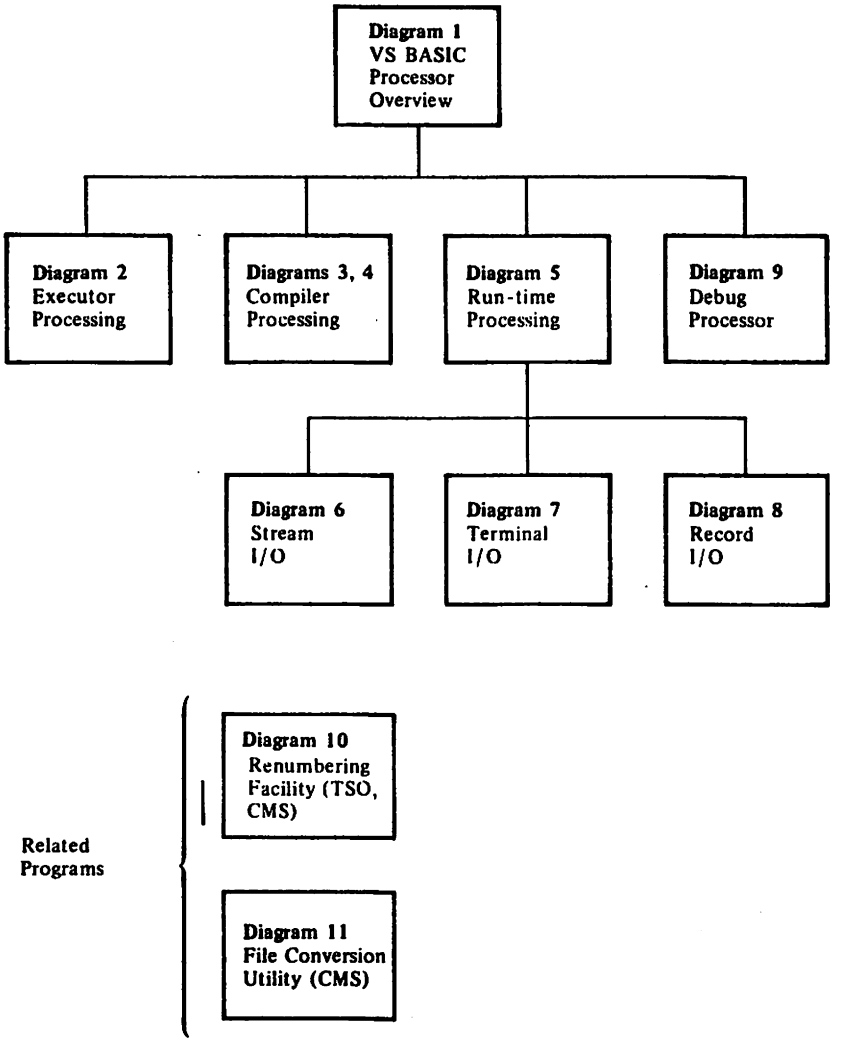
The utility can now begin processing the record. Data is obtained beginning at the seventh byte of the record. If the data is character, it is moved from the input buffer to the output buffer one byte at a time. Character strings that are longer than the output buffer will be split and placed in succeeding records. If the data is arithmetic, it is loaded into floating-point register 2 (in either single or double precision). The address of the corresponding conversion format is loaded into general register 2 and the run-time conversion routine (ICDKCNVT) is called to convert the data. If the converted string is too long for the output record, it is not split. However, a varying record that is shorter than the maximum record possible is placed in the output record. The RDW is calculated before the PUT routine is called to write the record. When the input record is completely converted, the QSAM GET routine obtains the next record in the file. Processing continues until all the records are converted. When a file is completely converted, it is closed and, the next file in the parameter list is obtained and converted. Processes stops when the last file has been processed.

THE HIPO TECHNIQUE

HIPO is a method for graphically describing the internal functions of a program without regard for the way in which the functions are implemented or for the physical organization of the program. A HIPO package contains a visual table of contents and a set of method of operation diagrams illustrating the functions of a program, in this case the VS BASIC processor. The visual table of contents shows the contents of each diagram and how it is related to the others in the set. The method of operation diagrams are grouped by function.

The method of operation diagrams themselves are divided into four distinct areas of information: input, process, output, and extended description. The input information, on the left side of the diagram, describes the input to the process or function being described. The process information, the central portion of the diagram, describes processes that make up the function. The output information, on the right side of the diagram, illustrates the output from the process. The extended description information below the diagram is used to provide additional detail or to outline how the function was implemented. This section also contains references to the module that performs all or part of the function involved and any references within the rest of the PLM where additional information may be found.

VS BASIC Processor: Method of Operation Contents



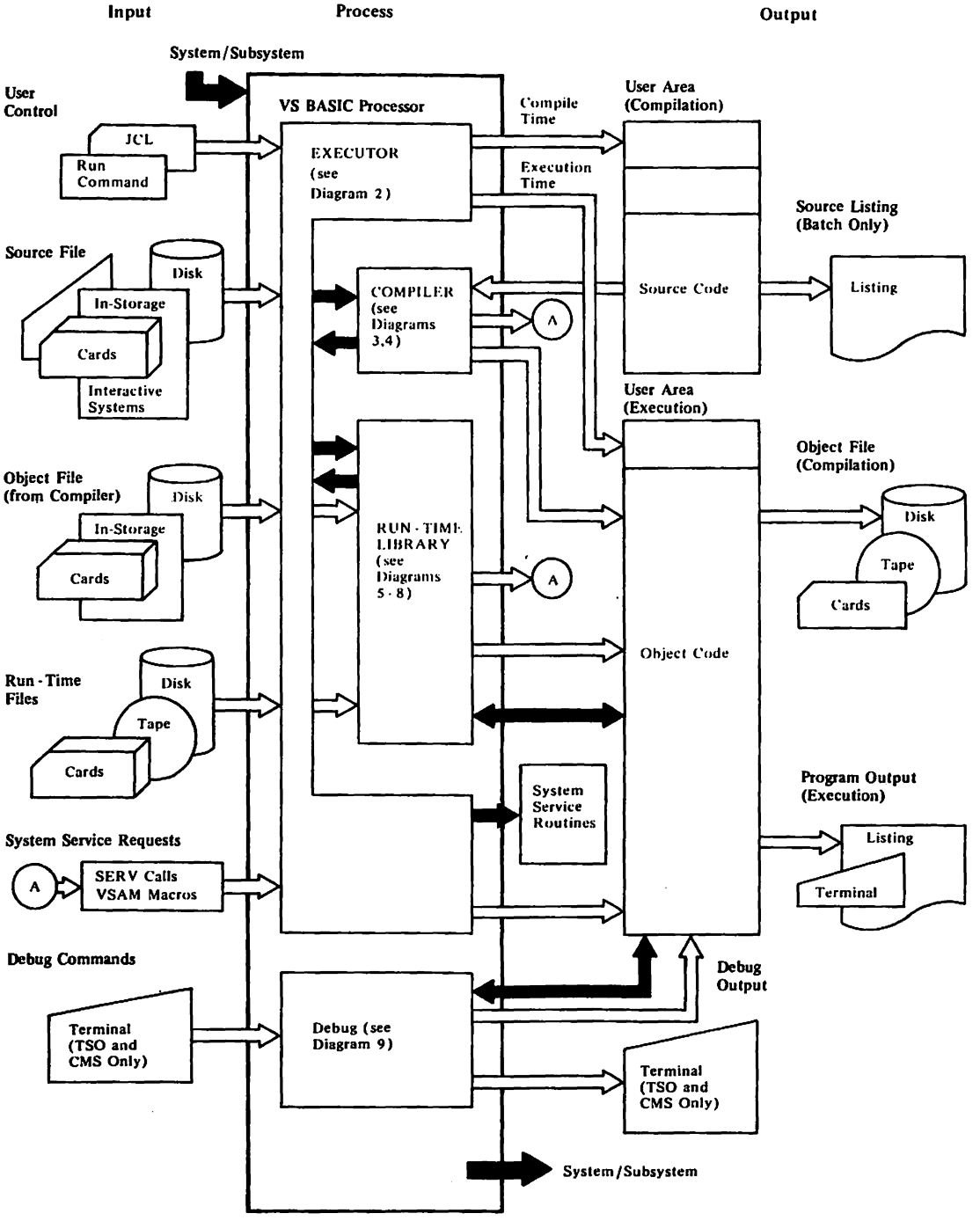


Diagram 1. VS BASIC Processor Overview

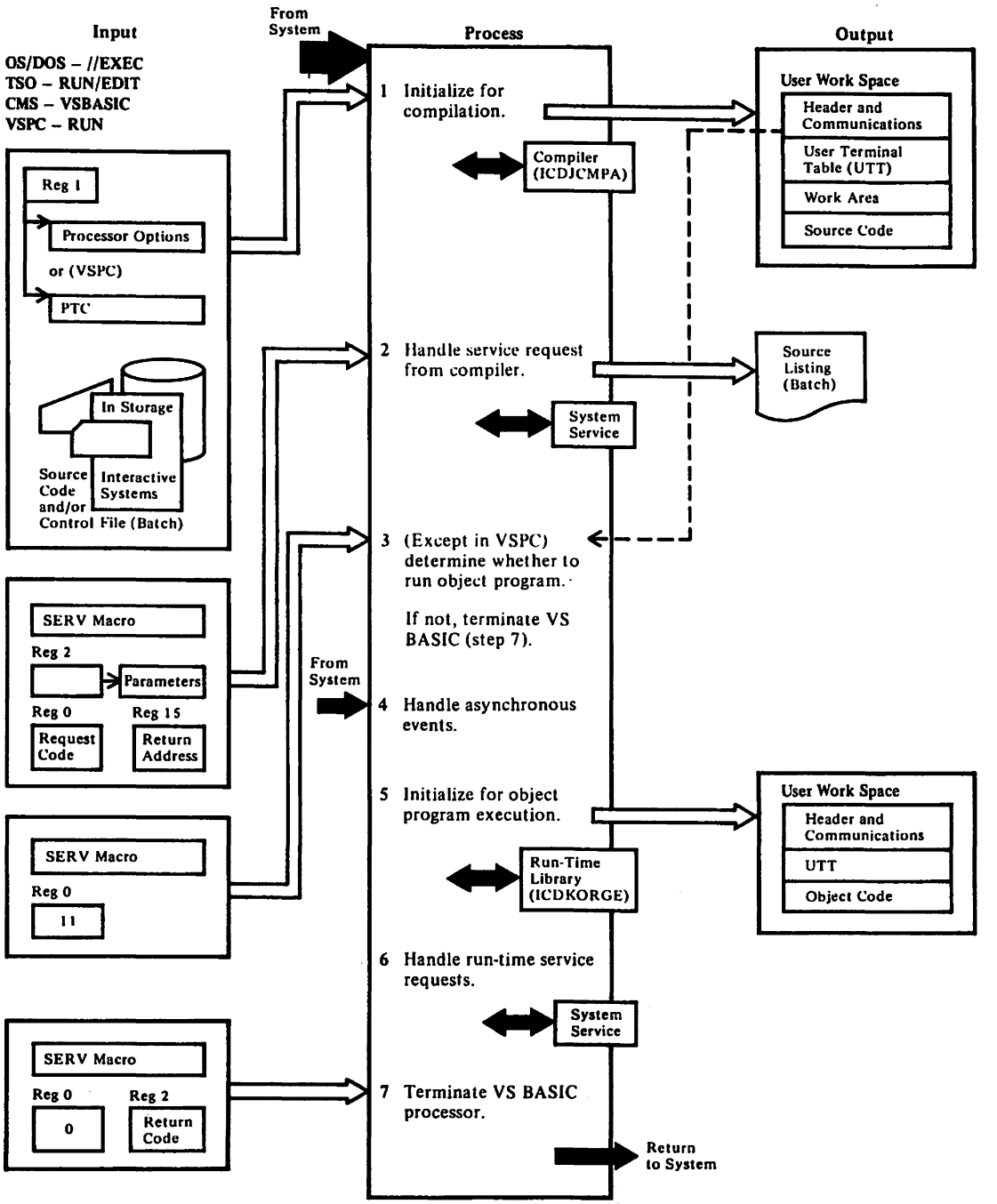


Diagram 2 (Part 1 of 3). Executor Processing

ICDxEXEC (x is P, Q, W, Y, or Z depending on the system. See Section 1 for description of the system-unique executors.)

1 The executor obtains a work area for its use and determines the initialization required from user-request options and/or the system environment. The interrupt-handling environment is set up depending on the system. The VSPC subsystem, however, provides this function for VS BASIC. The user's workspace is obtained and initialized. The source program is set up at the end of the user's workspace in the following edited format. Trailing blanks are removed; character-count and statement-end indicators are inserted. The end of program is indicated by a character count of one (X'01'). While setting up the source to be compiled, the executor determines the number of statements and the number of bytes in the source program. The information is placed in the User Terminal Table (UTT) in the workspace. The compiler is loaded, if necessary, and given control after initialization. (In CMS and VSPC, the compiler was loaded with the executor. Also, the CMS executor issues a GETMAIN if the DEBUG processor is needed.)

2 The executor handles service requests from the compiler for writing error messages, releasing storage, and terminating compilation. Requests are via the SERV macro instruction. The SERV number in register 0 indicates the particular service required. The executor relays the request to the system and, if required, returns system information.

All service numbers (0-26) and their functions are listed in Section 3 with the executor entry points. [ICDxEXEC,SVCRET]

3 After normal end of compilation, the compiler issues SERV request (11) to the executor. (In TSO, the executor then deletes the compiler. In CMS, it overlays the compiler if necessary.) If STORE option is in effect, the executor writes out the workspace area, which is stored as the object file. If mode is RUN, the system/subsystem branches to the object code through ICDKORGE. (See step 5.)

If severe source errors were noted during compilation, the compiler issues SERV request (0) to terminate the VS BASIC processor. See step 7. [ICDxEXEC]

4 The executor handles interrupts for program checks that occur in compiling or execution. It passes control through the system to the error handler in the compiler or run-time library. If a program check occurs in the executor itself, processing terminates without attempting error recovery.

During execution, asynchronous interrupts that result from system services are handled by the corresponding service routine. See step 6. [ICDxEXEC]

5 Before execution of an object program, the executor ensures the VS BASIC workspace is initialized with execution values in the communication area and in the UTT. If a just-compiled program is to be executed, the workspace from compilation is reinitialized. If an object program which had been stored is to be executed, the workspace must be obtained and set up. (The VSPC subsystem loads the workspace before calling VS BASIC.) The executor also checks for extended precision and for the CHAIN parameter. If CHAIN was used, the executor uses system/subsystem services to move the chained program to the workspace.

If workspace relocation is necessary (that is, the location is different from the compiled address), the executor performs the relocation. The executor gives control to the initialization routines of the run-time library (ICDKORGE). See Diagram 5. [ICDxEXEC]

6 While the object program executes, its requirements for files, I/O, time of day, or program chaining all result in a SERV macro instruction. The object code contains a call to a library routine which, in turn, issues the SERV call to the executor for the particular request. The executor's service routines generally interpret and relay the information for the request to the system/subsystem. After the system/subsystem returns control to VS BASIC, the executor analyzes results of the service request and sets indicators for the run-time library. Then, it returns control to the library routine. For a listing of service routines, see Section 3: Program Organization. [ICDxEXEC]

7 If a program check in the VS BASIC processor occurs, the executor is called via SERV (23). Any data in the buffer is written and an error message is issued. SERV (23) terminates VS BASIC.

For normal termination, the executor is called via SERV (0). Any remaining data is written out by the executor or (if VSPC) by the subsystem. Trailer messages are written if batch processing is in effect. All system resources are released. Return is to the system/subsystem. [ICDxEXEC]

Diagram 2 (Part 3 of 3). Executor Processing

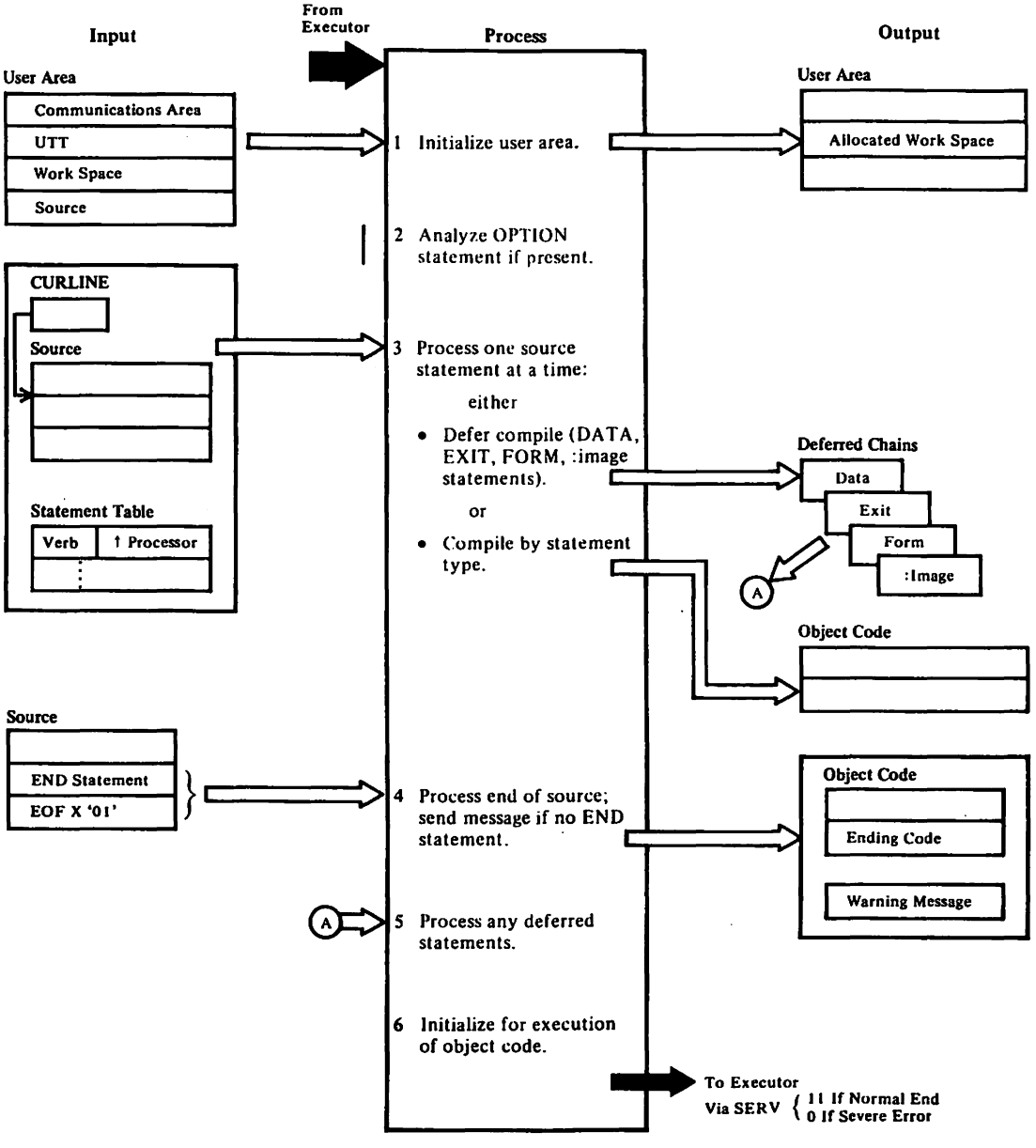


Diagram 3 (Part 1 of 3). Compiler Processing

ICDJCMPA

1 The VS BASIC workspace area not previously allocated by the executor is allocated in 4K blocks. The communications region is initialized. The current Release number is set up and, if the first statement is OPTION, switches are set up accordingly. The absolute save area, and the variable/constant areas (VARCON and VARCON1) are established. Initial code is placed in OBJAREA. The object code overflow area (4K bytes) is set up. Long/short precision switches are set, and data areas are initialized. Control is passed to ICDJCTL for statement processing. [ICDJCMPA]

ICDJNUCL

2 The statement-processing control routine (ICDJCTL) receives a pointer to the character count at the start of the first source line. ICDJCTL handles initial processing for the line and then determines the statement type from the verb. It sets the SRCPTR in register one to the end of the keyword.

Whenever a DATA, EXIT, FORM, or :image statement occurs ICDJCTL passes control to the ICDJDIMG routine for initial deferred-statement processing. The statement pointer is placed on the appropriate LINETAB chain. Processing for the next statement is begun via the ICDJCEND routine. For all other statement types, the verb table (VBTAB) is searched for the address of the routine to produce the corresponding object code. That routine is given control.

If an error occurs, the statement processing routine requests error handling by calling ICDJERR. Compilation then either terminates or the statement processing routine completes its operation after an error message is printed.

Before passing control to ICDJCEND for end of statement, the statement processor updates SRCPTR to the character count preceding the next statement. ICDJCEND returns to ICDJCTL to begin compiling the next statement.
[ICDJCTL,ICDJDIMG,ICDJCEND]

3 If the end of the source is reached without processing an END statement, ICDJCTL calls ICDJERR to send a warning message. This condition is noted when, on entry to ICDJCTL, the lines already processed (CNOLINE) is equal to the total lines of source (NOLINE). In this case, ICDJCTL passes control to ICDJEND (module ICDJDEFR) to supply the object program's ending code as if an END statement were being processed. The code produced is a branch to library routine ICDKRUNX (module ICDKERR) for normal execution end. [ICDJCTL,ICDJEND]

ICDJDEFR

4 After end of source has been processed, the chains of deferred statements are processed. Information from the LINPTRS table is used to set the values of SRCPTR, OBJREG, and CURLINE. The routine which handles each chain sets up pointers for the next chain routine. The last chain processing routine (ICDJDATA) passes control to the compiler's run-time initialization routine (ICDJRUNA). Control passes via ICDJCEND.
[ICDJCDEF,ICDJIMAG,ICDJFORM,ICDJEXIT,ICDJDATA]

ICDJRUNA

5 ICDJRUNA checks for compilation errors and checks for an incomplete FOR loop or DEF. If any of these conditions exist, ICDJERR is called.

Diagram 3 (Part 2 of 3). Compiler Processing

Otherwise, ICDJRUNA checks whether VARCON2 was allocated. If it was the allocated VARCON2 area is moved to the end of the object code. If TEST is in effect, several tables in PRGA are saved and moved next to VARCON2. ICDJRUNA then calls the executor via SERV request (5) to release unneeded storage. The communication area and registers are set up for execution. (Compile-only area in PRG is replaced by run-time area.) Compiler exit is via SERV request (11) to the executor. [ICDJRUNA,ICDxEXEC]

Diagram 3 (Part 3 of 3). Compiler Processing

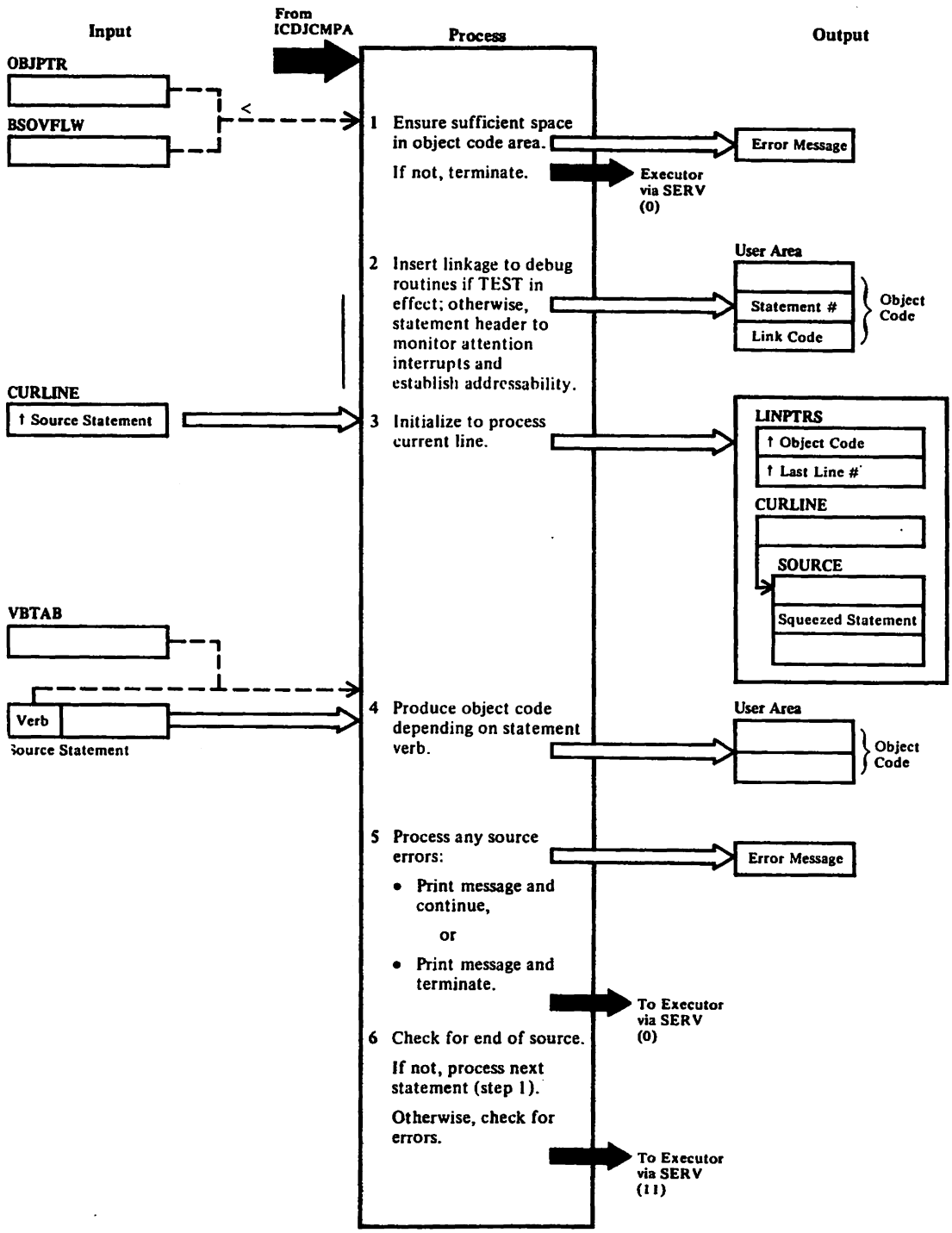


Diagram 4 (Part 1 of 3). Compiler -- Statement Processing

ICDJNUCL

1 ICDJCTL checks whether additional space for the object code is required before processing the current statement. The condition exists if the value of object code pointer (OBJPTR) has already reached the address of the overflow area (BSOVFLW). If the initial area has been exceeded but the overflow area is still available, the object code area is expanded. If the overflow area is exceeded, compilation terminates through ICDJERR. ICDJERR issues SERV request (0) to the executor, and a size-error message is sent. [ICDJCTL]

Statement Processor Modules

2 If the TEST option is in effect, both the statement number and linkage to the DEBUG routines is placed in the object area before the statement itself. Otherwise, code is generated to monitor attention interrupts and establish addressability.

ICDJNUCL

3 The LINTPRS entry is made by ICDJCTL for this statement. The entry includes the source line number and the address where the object code will be placed. Nonsignificant blanks, tab characters, or remarks (REM) in the source line are removed. The error exit is taken to ICDJERRT in ICDJERR either if a line number is invalid or if there is no line number. [ICDJCTL,ICDJLINE]

4 ICDJCTL does a binary search of VBTAB for the address of the statement processor for the current verb. If the statement does not have a verb, LET is assumed. Remarks (REMs) result in processing of the next statement.

If a DATA, EXIT, FORM, or :(image) statement is found, a LINTAB entry is added to the appropriate chain by the corresponding routine in ICDJNUCL. The statement is not compiled until the END statement is processed.

For other statements, ICDJCTL calls the statement processor listed in VBTAB. (The modules are listed below with respective verbs. Section 3 lists individual entry points within the modules.) Statement processing includes: checking syntax, evaluating statement elements, checking for any required conditions, determining run-time links required, and generating the object code. Function references in a statement cause the intrinsic function to be provided as in-line code or as a run-time library linkage. The ICDJFUTS module generates the code and builds any argument lists required. Appendix A lists the object code for each statement type. [ICDJCTL,ICDJDIMG]

Diagram 4 (Part 2 of 3). Compiler -- Statement Processing

The modules which contain statement processors are:

MODULE NAME	STATEMENT
ICDJIOVB	CLOSE, GET, INPUT, INPUT FROM, PAUSE, PRINT (PRINT, FORM, Image), PRINT TO, PUT, READ, RESET, RESTORE
ICDJMATV	MAT
ICDJNUC2	FOR, LET
ICDJNUC3	NEXT
ICDJNUC4	GOSUB, GOTO
ICDJNUC5	IF
ICDJUSFN	DEF, FNEND, RETURN
ICDJVERB	CHAIN, DIM, STOP, USE, ON
ICDJVREC	VSAM I/O

ICDJERR

5 The statement processing routines call ICDJERRP to handle syntax errors. More serious errors invoke ICDJERRN or ICDJERRT, which issue a service call (0) to the executor to terminate compilation and issue the error message with the line number. Five bytes of source code may be printed. Really fatal problems stop immediately via ICDJERRS, which issues service call 23. [ICDJERRN, ICDJERRP, ICDJERRT, ICDJERRS]

Diagram 4 (Part 3 of 3). Compiler -- Statement Processing

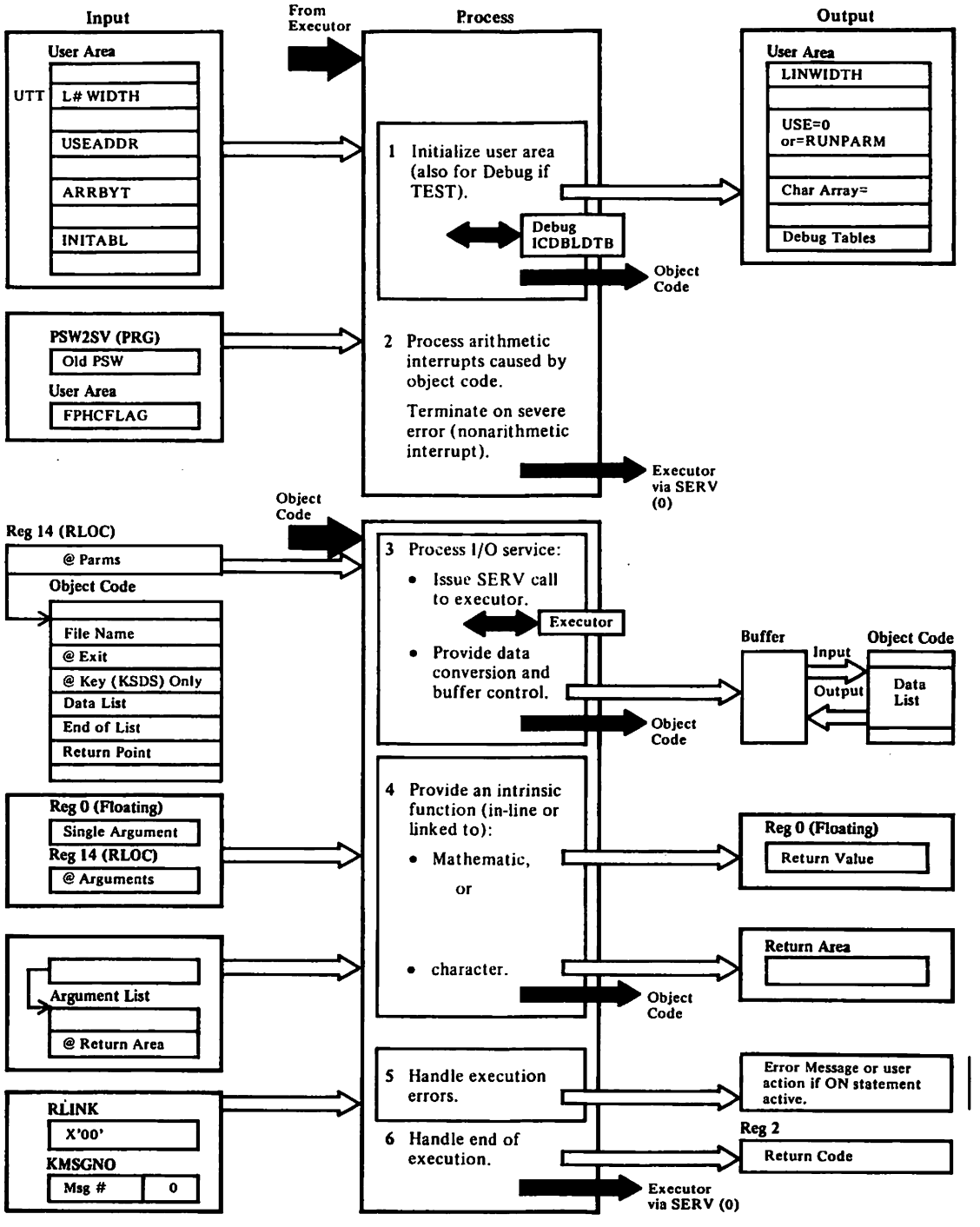


Diagram 5 (Part 1 of 3). Run-time Processing

ICDKORGE

1 A check is made to ensure the object program is compatible with the run-time library; if not, execution is terminated with a call to SVC23. The current line width (LINWIDTH) is set using the L#WIDTH value from the UTT.

If the object program is chained, ICDKORGE sets the USE variable with the value provided by the chaining program. USEADDR points to the USE statement, and USEPARM points to the variable. The value provided by the chaining program is contained in RUNPARM in PRG. If RUNPARM is null, the chained variable is set with blanks.

To get storage needed for arrays, debug tables, a temporary DFT, or workspace, ICDxEXEC is called via SERV request (6). Array storage is obtained if ARRBYT is nonzero. INITABL tells whether arrays are character or not. For character arrays, storage is initialized with blanks; for numeric arrays, storage is set to zero. Attention monitoring code is moved to the beginning of PRG. If TEST is in effect, Debug tables are obtained and initialized by ICDBLDTB. Control is then passed to the object code. [ICDKORGE,ICDBLDTB]

ICDKINTP, ICDKERR

2 If an arithmetic interrupt is for exponent overflow, underflow, or zero divide, ICDKINTP requests an error message produced by ICDKERR. The floating-point register used by the instruction is set to maximum or zero, and SERV request (8) is issued to the executor for return to object program execution. If any other arithmetic interrupt occurs, ICDKINTP calls ICDKERR, which issues an error message and SERV request (23) for program termination. [ICDKINTP,ICDKERRR]

NOTE: If an interrupt was not caused by an ATTN, the executor returns to the library for abnormal termination. [ICDKERRT]

OBJECT CODE CALLS: The library routine needed for a service or function was determined by the compiler from conditions in the source. For example, file attributes determine whether object code linkage is to a record stream, or terminal I/O routine.

ICDKIOVB-Stream I/O
ICDKVIOR-Record I/O
ICDKINPT, ICDKPRNT-Terminal I/O

3 The library routines do file checking in preparation for an open, close, reset, or I/O. Before any I/O operation, the library routines test IOSW flag for any attempt to do nested I/O. If the switch is already set to one, object code execution is terminated. Otherwise, the library routines issue the SERV request with the code in register 0 specifying the required operation. The executor relays the request to the system/subsystem and returns to the library routine after the operation. (If the workspace has been relocated during the service call, the executor adjusts accordingly addresses and registers used for addressing.) The library routine returns information to the object program. Depending on the request, data may be converted, formatted, and moved. Array element manipulation is handled.

Diagram 5 (Part 2 of 3). Run-time Processing

ICDKxSUB (x is D, G, or S)

4 The in-line functions require only one argument. These are ABS, RAD, SGN, DEG, INT, FAH, CEN. CNT requires no arguments. The math functions with one argument include ACS, ASN, ATN, COS, COT, CSC, DET, EXP, HCS, HSN, HTN, LGT, LOG, LTW, SEC, SIN, SQR, TAN. For math functions with a list of arguments, the first argument is the number of elements in the list. These are: RND, MAX, MIN, TIM, CPU, SUM, PRD, DOT. Character functions have an argument list, with the last element being a pointer to the return value area. These are: DAT, STR, JDY, CLK, IDX, KPS, KLN, RLN, CHR, NUM, and LEN.

ICDKERR

5 The library routines receive the message number for an error condition in Register 0. If the user has not specified he wants control by means of the appropriate ON statement or I/O error clause, the error routine issues the message and a SERV0 request for normal termination. Severe system errors always terminates with SERV request 23. [ICDKERRR,ICDKERRS,ICDKERRT]

ICDKERR

6 If a CHAIN statement is issued by the object code, the program is terminated and a service request (16) is issued to the executor to begin executing the chained program. [ICDKCHN]

Normal end of object program results in a branch to ICDKRUNX, which issues service request (0) to end VS BASIC processing. [ICDKRUNY,ICDKRUNX]

Diagram 5 (Part 3 of 3). Run-time Processing

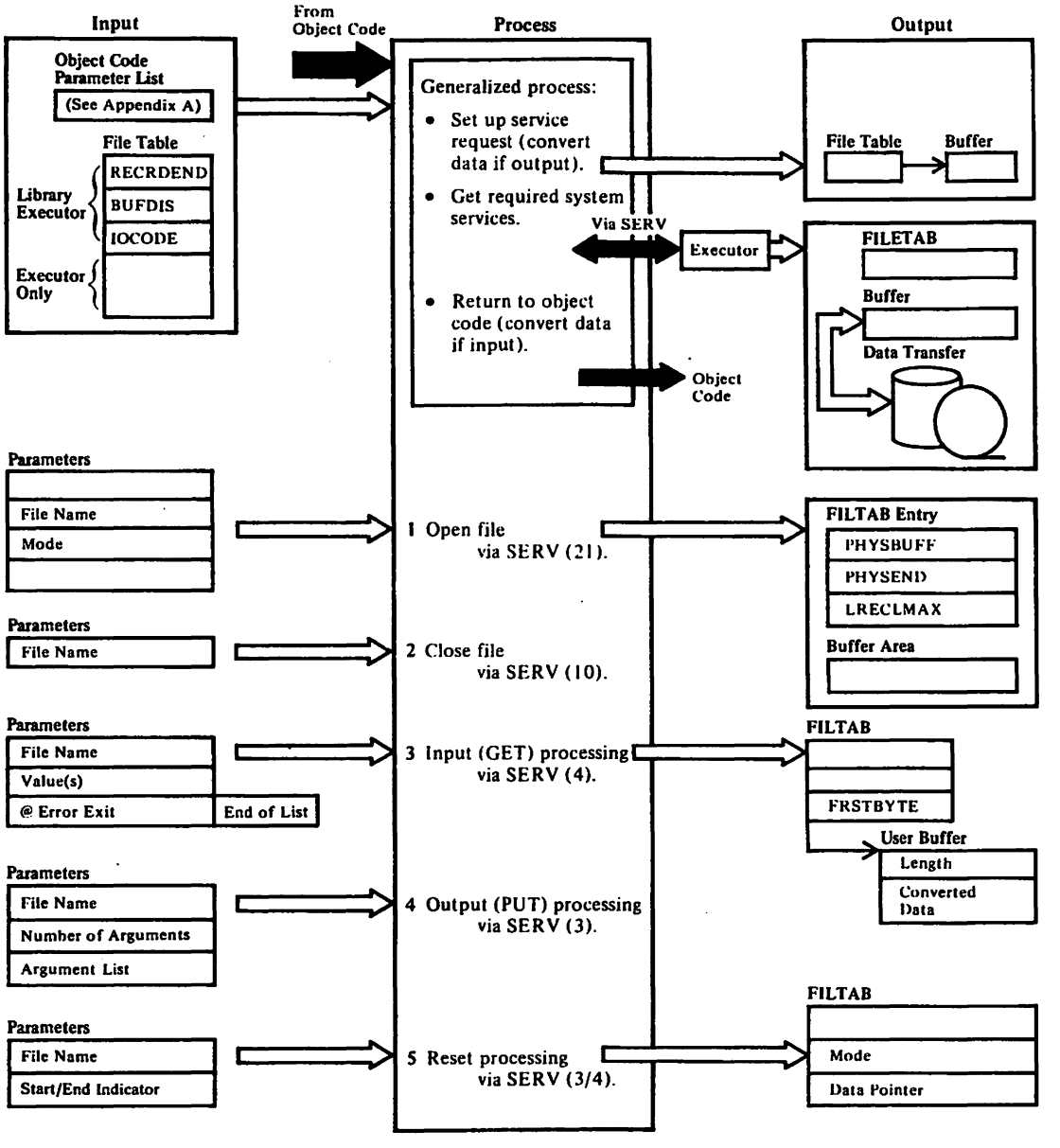


Diagram 6 (Part 1 of 2). Run-time Stream I/O

ICDKIOVB

1 Open-The file scan routine (ICDKFSCN) is called to check whether the file is already open. The ICDKOPEN routine sets up a temporary DFT and issues SERV (21). The executor obtains storage (except in CMS) and sets up the permanent DFT and the buffer. Depending on the system, other control blocks may be required. [ICDKOPEN]

2 Close-The ICDKCLOS routine uses the file scan module ICDKFSCN to get the correct file table entry. Then, if the output buffer has data in it, ICDKCLOS issues SERV (10) to write output and then close the file. Otherwise, SERV (22) is issued to close the file. If an error occurs ICDKERRS terminates execution. [ICDKCLOS]

3 Input (GET)-A file scan is done by calling ICDKFSCN. If the file has not been opened, the ICDKOPN1 routine is called. The file is verified, and if it is not available for input, ICDKERRS prints a VS BASIC error message and terminates execution. Otherwise, the buffer is scanned for the next field and, if necessary, SERV (4) is issued for another record. When the library GET routine regains control, it calls the I/O conversion module ICDKETOF to move character data or to move and convert numeric data. If there are array arguments, the elements are converted singly. Input processing continues until the end of the parameter list is reached unless an error has occurred. Control is returned to the next instruction in the object code. [ICDKGET]

4 Output (PUT)-A file scan is done by calling ICDKFSCN. If the file has not been opened the ICDKOPN1 routine is called. If the file is not available for output, a VS BASIC error message is issued and execution terminates. Arithmetic data is converted by the ICDKCNVT routine. Array elements are handled singly. The executor is called via SERV (3). Output processing continues until the end the parameter list or an error occurs. Control returns to the next instruction or to the exit address in the object code. [ICDKPUT]

5 Reset at end-ICDKFSCN is called to check the file status. For a file previously opened for input, the reset to end is done by reopening the file in output mode via SERV (21). The data pointer is set to the end of existing data. Control returns to the object code.

If the file was open for output already, control returns to the object code immediately.

Reset at beginning-The file scan routine is called to check file status. A file open for input is repositioned to the beginning by a service request (3), if the file was open in output mode. Service request (4) is used if the file is open in input mode. [ICDKRSET]

Diagram 6 (Part 2 of 2). Run-time Stream I/O

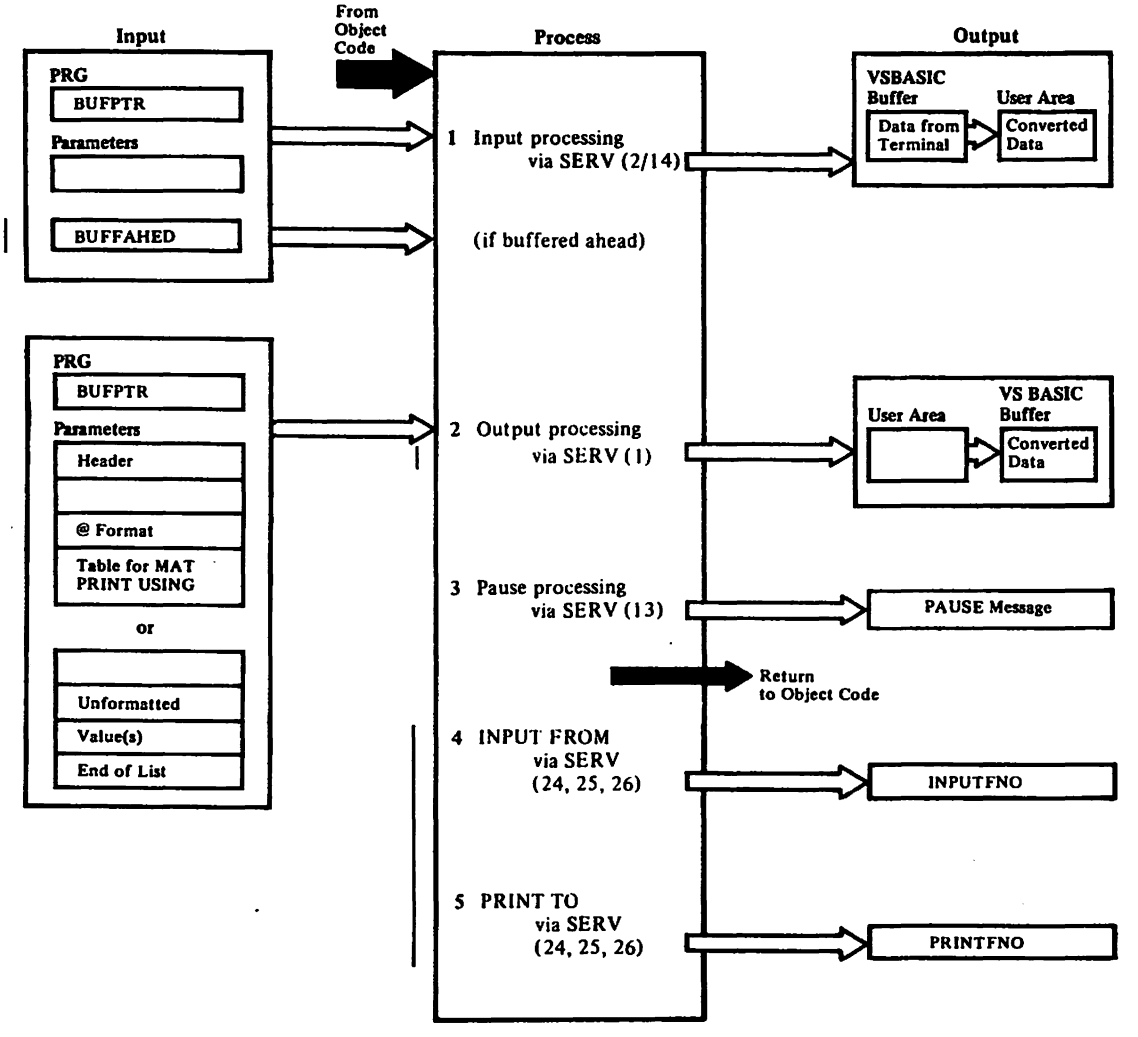


Diagram 7 (Part 1 of 2). Run-time Terminal I/O

ICDKINPT, ICDKPRNT (Unit Record)

1 Input-The routine sets the CNT field to zero initially. The subroutine BUFFCHK is used to obtain INPUT data. If data is available within BUFFAHED, a 'line' is obtained from there and &BUFF is decremented. If data is not available in BUFFAHED, SERV(2) is issued to read a record from the terminal or, if INPUT FROM a filename is active, SERV(24) is issued to read a record from the file identified by INPUT FROM. The record obtained is scanned for semicolons (;), which are counted and the number placed in &BUFF. If &BUFF is non-zero, the excess 'lines' are moved to BUFFAHED. If terminal input is not accepted, SERV(14) is issued to reread the data. (The latter service request results in program termination in a batch environment, or if INPUT FROM a file is active.) The ARGSAV entries are decoded. If there is an array, the ARYDSC is located, and the correct number of elements to be read (row by row) is determined. Numeric data is converted by the ICDKETOF routine (module ICDKETOF). Character data is converted and/or moved by ICDKETF2. [ICDKINPT]

2 Output-The FMTFLG is set to X'00' for unformatted, X'01' for image, and to X'02' for FORM requests. Arrays are processed row by row and as if each element were a single variable. Arithmetic expressions, in an array or not, are processed as positive values, with the SIGN flag indicating positive or negative sign. ICDKCNVT converts the number to EBCDIC. If there is a FORM, the output routine ensures it is an arithmetic format. ICDKPRNT calls ICDKPLIN to move character expressions or blanks (for null string) to the internal print buffer. SKIP, POS, and X controls are processed. ICDKPRNT calls ICDKTOUT which issues SERV (1) to perform the output, or SERV(24) if PRINT TO filename is active. Pause statements are handled by ICDKPRNT, which issues the SERV(13). [ICDKPRNT, ICDKTOUT]

3 PRINT TO/INPUT FROM - Terminal files are handled by ICDKTIO, an entry point within ICDKPRNT. It maintains the two bytes within PRG (PRINTFNO and INPUTFNO) that control which PRINT TO/INPUT FROM files are active (see ICDKTOUT and ICDKINPT above). If the referenced file is not open, ICDKVTIO, an entry point within ICDKVIOR, is called to activate that file.

Diagram 7 (Part 2 of 2). Run-time Terminal I/O

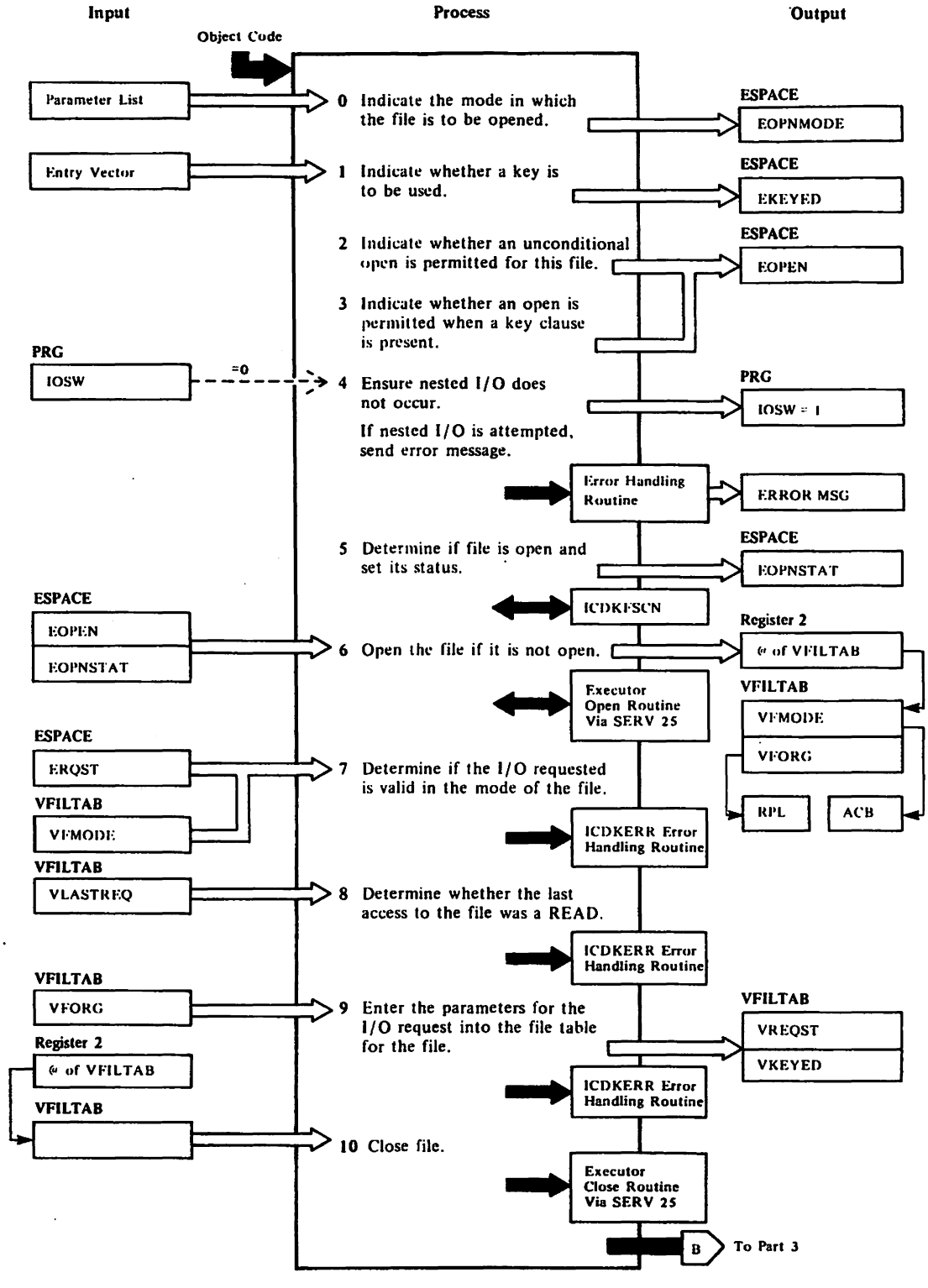


Diagram 8 (Part 1 of 6). Run-time Record I/O

Record-oriented I/O processing is controlled by module ICDKVIOR, which contains an entry point for each type of I/O request. Each of these entry points is followed by five bytes which describe the parameters of a particular request. These fields (which are moved into ESPACE), include:

- Mode in which the file is to be opened (EOPNMODE).
- The request code (EREQST).
- Mode in which request would be invalid (ERMODE).
- Length of parameter list (EPLINCR).
- Placement of key in parameter list (EKEYINCR).

These five bytes are followed by a variable number of bytes which define the sequence (and the number) of ICDKVIOR steps required for this request. For each request, the information is as follows.

I/O STATEMENT	ENTRY POINT	SEQUENCE OF STEPS EXECUTED
OPEN FILE	ICDKVOPN	0, 2, 4, 5, 13, 6, 19, 24
READ FILE	ICDKVRD	1, 2, 4, 5, 6, 7, 9, 11, 12, 13, 18, 14, 15, 22, 24
REREAD FILE	ICDKVRRD	1, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14, 15, 18, 22, 24
RESET FILE	ICDKVRST	1, 3, 4, 5, 6, 7, 9, 12, 13, 18, 14, 19, 24
WRITE FILE	ICDKVWRT	1, 2, 4, 5, 6, 7, 9, 11, 13, 15, 21, 22, 16, 17, 18, 14, 24
REWRITE FILE	ICDKVRWR	1, 3, 4, 5, 6, 7, 8, 9, 12, 13, 18, 14, 20, 15, 21, 11, 22, 16, 17, 18, 14, 19, 24
DELETE FILE	ICDKVDEL	1, 2, 4, 5, 6, 7, 9, 12, 13, 18, 14, 19, 24
CLOSE FILE	ICDKVCLS	1, 3, 4, 5, 6, 13, 10, 19, 24
Program End	ICDKVEND	10
Implicit open of terminal file	ICDKVTIO	0, 2, 4, 5, 23, 6

ICDKVIOR

Open: Whether the file is to be opened for input, output, all, and/or hold is determined from the parameter list [step 0]. The EOPNMODE flags are set accordingly. The open-is-valid indicator (EOPEN) and the IOSW are set on [steps 2,4]. The file table for this user is scanned to determine the file status; open, openable, or file table full [step 5].

Any exit argument specified is placed in VEXITDISP [step 13]. The SERV (25) is issued to open the file, provided the file is not currently open [step 6]. Upon return from the executor, register 2 points to the file table, VFILTAB. The library open routine returns to the object code.[ICDKVOPN]

Read: ICDKVRD processing checks the parameter list for a keyword and sets the EKEYED indicator accordingly [step 1]. It also sets EOPEN to show open is valid for this file [step 2]. The file scan routine, ICDKFSCN, is called to check this user's file table to determine whether

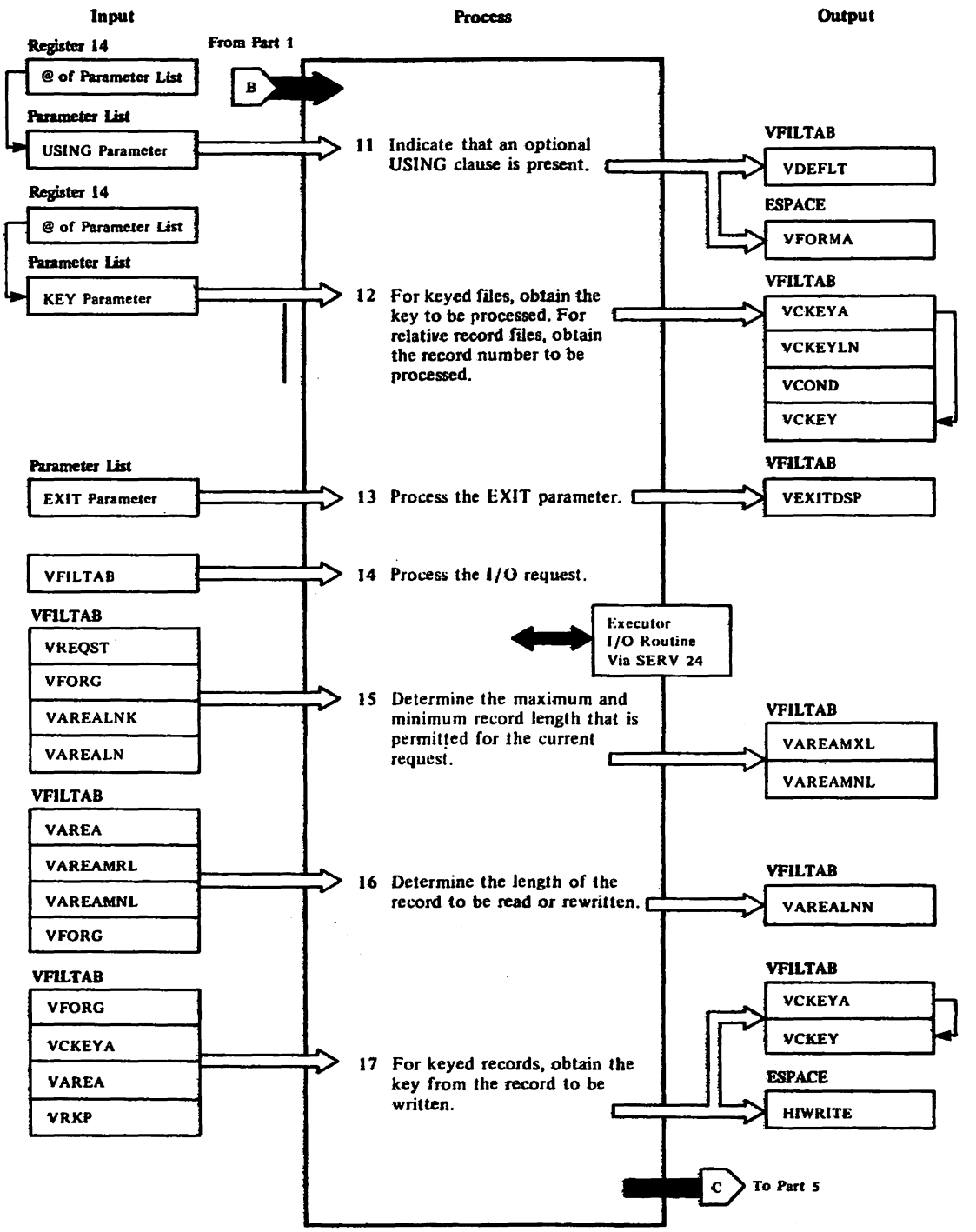


Diagram 8 (Part 3 of 6). Run-time Record I/O

the file is already open and whether the file table is already full [step 5]. If the file is not open, the library sends the open request to the executor [step 6]. After successful open, the library ensures that a read request is valid for the open mode of the file and that the file is KSDS if a keyword was specified [steps 7,9]. Any exit argument specified is placed in VEXITDSP [step 13]. If there was previously an I/O-error recovery attempt and the current read is sequential, the request cannot be filled [step 18]. In this case, an error message is issued and the error exit is taken. If there is no error and if the file was initially empty when opened, the file is closed and reopened [step 18]. Otherwise, the executor is called via SERV (24) [step 14].

If the read is successful, the minimum and maximum record lengths are set by the library [step 15]. If a list was read, the items are processed using FORM specifications or default data types and lengths [step 22]. The library returns to the object code unless an error exit was required [step 24]. [ICDKVRD]

Reread: EKEYED is set to show that no keyword is present. The EOPEN flag is set to show that open is invalid [step 3]. ICDKFSCN is called to scan the user's file table and determine the file status [step 5]. The library ensures the validity of the request; that is, the open mode must be compatible with the request, and the last access must have been a read [steps 7-9]. If the parameters include USING, the form indicator is set. Otherwise, indicators for defaults are set [step 11]. Any exit specified is noted in the file table [step 13]. SERV (24) is issued to the executor [step 14].

The library sets the maximum and minimum record lengths [step 15]. It also checks whether there was an I/O-error recovery attempt preceding this request. If there was, the error exit is taken [step 18]. Otherwise, any required list processing is performed using the form or default values [step 22]. Then, the library returns to the object code [step 24]. [ICDKRRD]

Reset: If the parameter list contains a key, the EKEYED indicator is set accordingly [step 1]. The library ensures the request is valid for the open mode and, if the request is keyed, that the file is a KSDS [step 3]. If the file is not open, reset does not open the file [step 6]. Reset does check for prior I/O error [step 18]. Any exit argument specified is placed in VEXITDISP [step 13]. SERV (24) is issued to call the executor. If a key was specified, it is used to fill the reset request; otherwise, the reset is to the beginning of the file. When the reset is completed, the library returns to the object code [step 24]. [ICDKVRST]

Write: The library sets EKEYED to no key present [step 1]. The file scan routine, ICDKFSCN, is called to determine the file status [step 5]. If the file is not open and can be opened, the executor is called via SERV (25) to process open [step 6]. If open was successful, the open mode is checked for compatibility with the write request [step 7]. Then, the parameter list is checked for a USING clause [step 11]. The default and form indicators are set accordingly. The exit indicator is set depending on whether the parameters include an exit [step 13]. Minimum and maximum record lengths are set [step 15]. The unused part of the record area is cleared [step 21]. List items are processed using form or default values [step 22]. The length of the record to be written is determined [step 16]. For a KSDS, the key is moved to the file table [step 17]. The executor is called via SERV (24) [step 14]. After the write, the library returns to the object code [step 24]. [ICDKVWRT]

Rewrite: If a key is provided, EKEYED is set on [step 1]. ICDKFSCN is called to determine the open status [step 5]. If the file is not open and can be opened, the executor is called [step 6]. The open mode is checked for compatibility with the rewrite request. If the last I/O request was not a read, the library ensures that the current request

Diagram 8 (Part 4 of 6). Run-time Record I/O

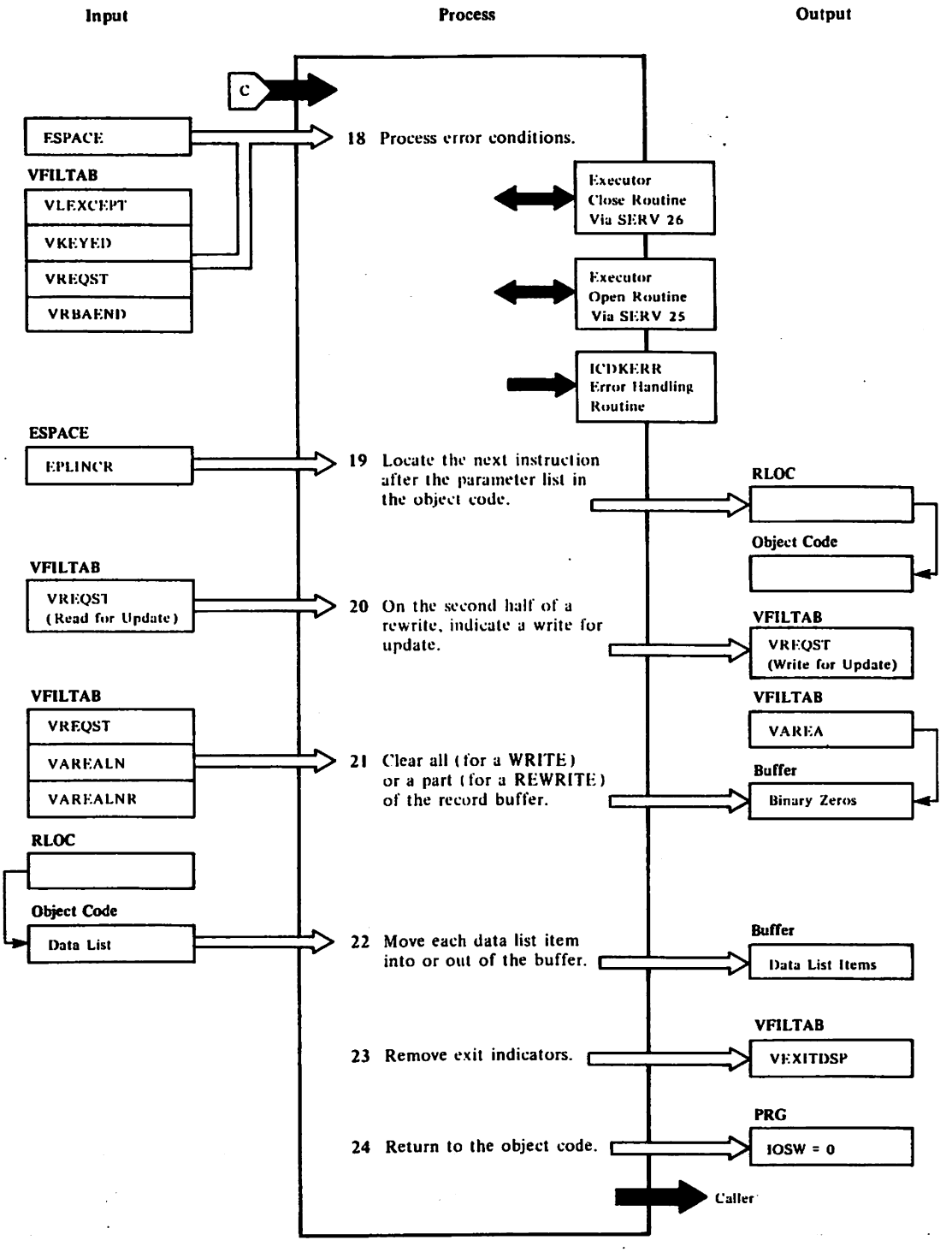


Diagram 8 (Part 5 of 6). Run-time Record I/O

specifies a key [step 8]. Also, if a key was specified, the file is checked for a KSDS [step 12]. The VEXITDSP field is set [step 13]. If there was an error recovery attempt on the previous request, this request is invalid (if nonkeyed). If the file was empty when opened and the current request is not a sequential write, the file is closed and reopened [step 18]. The executor is called via SERV (24) if the read for rewrite can be done [step 14]. The write-for-update indicator is set in the file table [step 20]. Minimum and maximum record lengths are set [step 15]. If the object code specified a USING, the form values are set; otherwise, defaults are indicated. Then, the list items are processed to reflect form or default values [steps 11,22]. The length of the record to be written is determined [step 16]. For a KSDS, the key is moved to the file table [step 17]. After calling the executor via SERV (24) to handle the write for update request, the library returns to the object code [steps 14,24]. [ICDKVRWR]

Delete: The object program parameter list is checked for a key [step 1]. The user's file table is scanned [step 5]. If the file is not open and can be opened, SERV (25) request is issued [step 6]. When the file is open, the library checks whether the request is compatible with the open mode and whether the file is a KSDS if a key was specified [step 7]. If no error conditions are found, the SERV (24) request is issued [step 14]. After the delete request is completed, the library returns to the object code [step 24]. [ICDKVDEL]

Close: If the file is not open when an implicit close is requested, the request is ignored. Any EXIT argument specified is placed in VEXITDISP [step 13]. If the file is open, the file is then closed via SERV (26) [step 10]. Before returning to the library, the executor removes file references in the file pointers table (TABLPTRS). The storage for the file table, record area, and any key area is released by the executor. Register 2 points to the temporary file table. When the library regains control, it locates the object instruction to regain control and passes control to it [step 24]. [ICDKVCLS]

Program End: SERV (26) is issued to close any open files [step 10]. [ICDKVEND,ICDKVCLS]

Implicit open for terminal file: The file is opened in the same way as ICDKVOPN except for two things: (1) the exit indicator is set to 'no exit', and (2) control is returned via register 4 to the library routine which invoked it (ICDKTIO). [ICDKVTIO]

Diagram 8 (Part 6 of 6). Run-time Record I/O

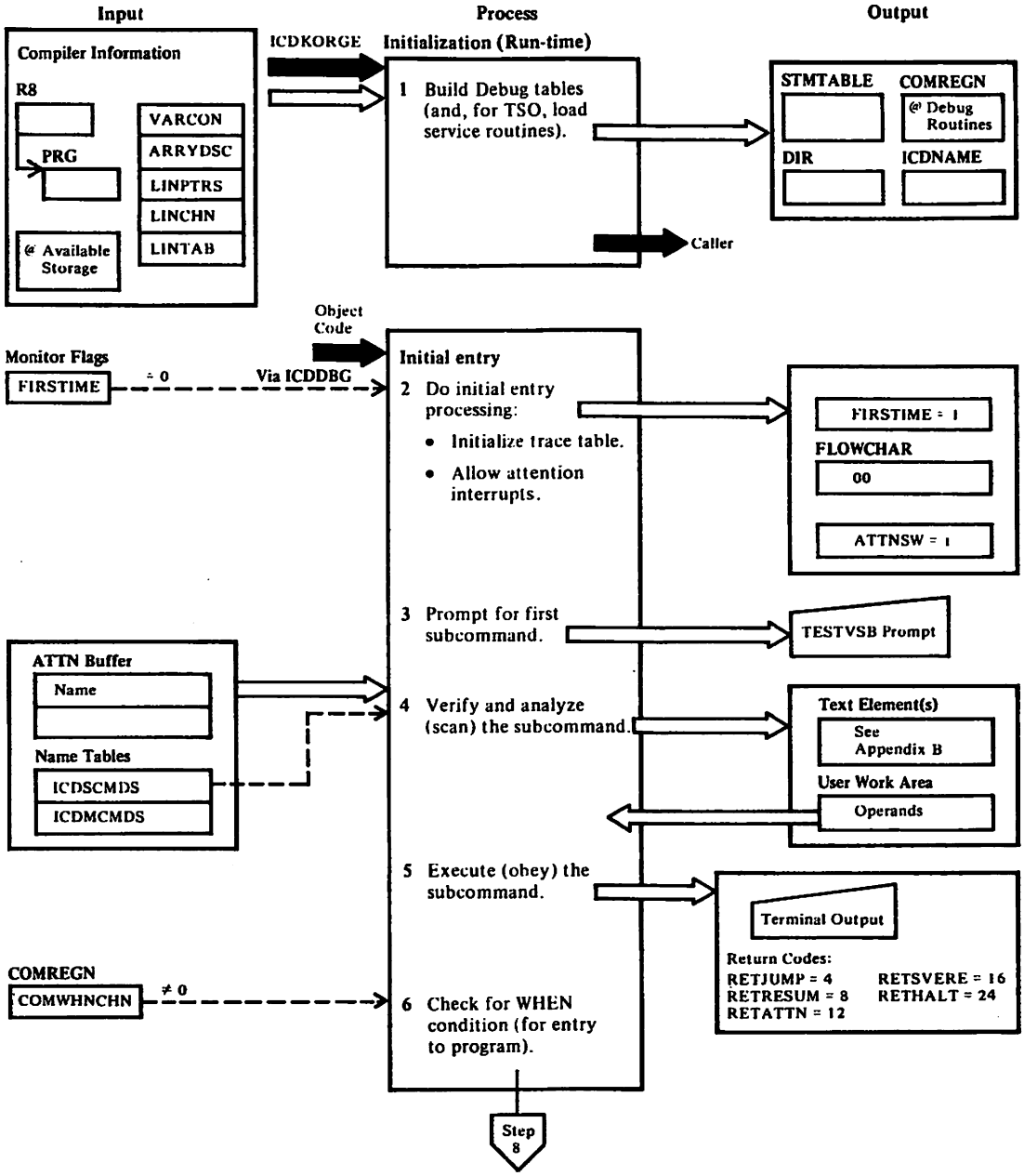


Diagram 9 (Part 1 of 5). Debug Processor

ICDBLDTB

1 Storage for the debug tables and a pointer in register 8 to PGM are passed to ICDBLDTB by ICDKORGE. ICDBLDTB builds a STMTABLE entry for each source statement using compiler tables LINTAB, LINPTRS, and LINCHN. The program-unit directory (DIR) is built from information in PGM. The symbol table (ICDNAME) is built from VARCON and ARRYDSC for the main program unit. The debug communications area (COMREGN) in ICDONITR is initialized. (ICDONITR, the debug monitor, is not reentrant since it is modified.) [ICDBLDTB]

ICDONITR

2 The Debug monitor ICDONITR receives control through ICDDBG, the object code interface. Monitor sets the FIRSTIME flag on when it is first entered. In this case, the monitor initializes the trace table FLOWCHAR to zero. It calls ICDCHAIN to get storage in subpool 77 for text elements used in subcommand processing. The monitor also sets the ATTNSW flag to 1 so that attention interrupts can be received.

Then, the scanobey routine within the monitor is called to control a series of subcommand promptings and executions. A series ends when a GO or RUN is entered. [ICDONITR,SCANOBEY]

ICDSCAN

3 The subcommand-scan dispatch routine ICDSCAN is called by scanobey. ICDSCAN calls for printing of the user prompt message, TESTVSB. The message is printed by ICDMODE contained in ICDMSSG. The scan dispatch routine receives the user's subcommand in the ATTN buffer. [ICDSCAN]

ICDDSCAN

4 This routine verifies the command by checking for the name in the ICDMCMDS or ICDSCMDS tables. ICDDSCAN determines the routine to handle the particular subcommand. ICDDSCAN calls the routine, which builds the text element(s). (See Appendix B for the contents of the text elements.) Also, any operand or syntax checking is performed. [ICDDSCAN]

ICDOBEY

5 The scanobey routine in the monitor calls ICDOBEY, which determines from the ICDOCMDS table the routine to execute the obey processing for the subcommand. The text elements produced during the scan phase are used in the obey execution. The obey routines use the following return codes:

CODE	MEANING
0	normal
4	jump required
8	resume processing
12	ATTN received
16	severe error
24	halt processing

[ICDOBEY]

ICDWNTST

6 ICDONITR calls the WHEN test routine to check for any WHEN conditions received during the initial entry. For example, if

Diagram 9 (Part 2 of 5). Debug Processor

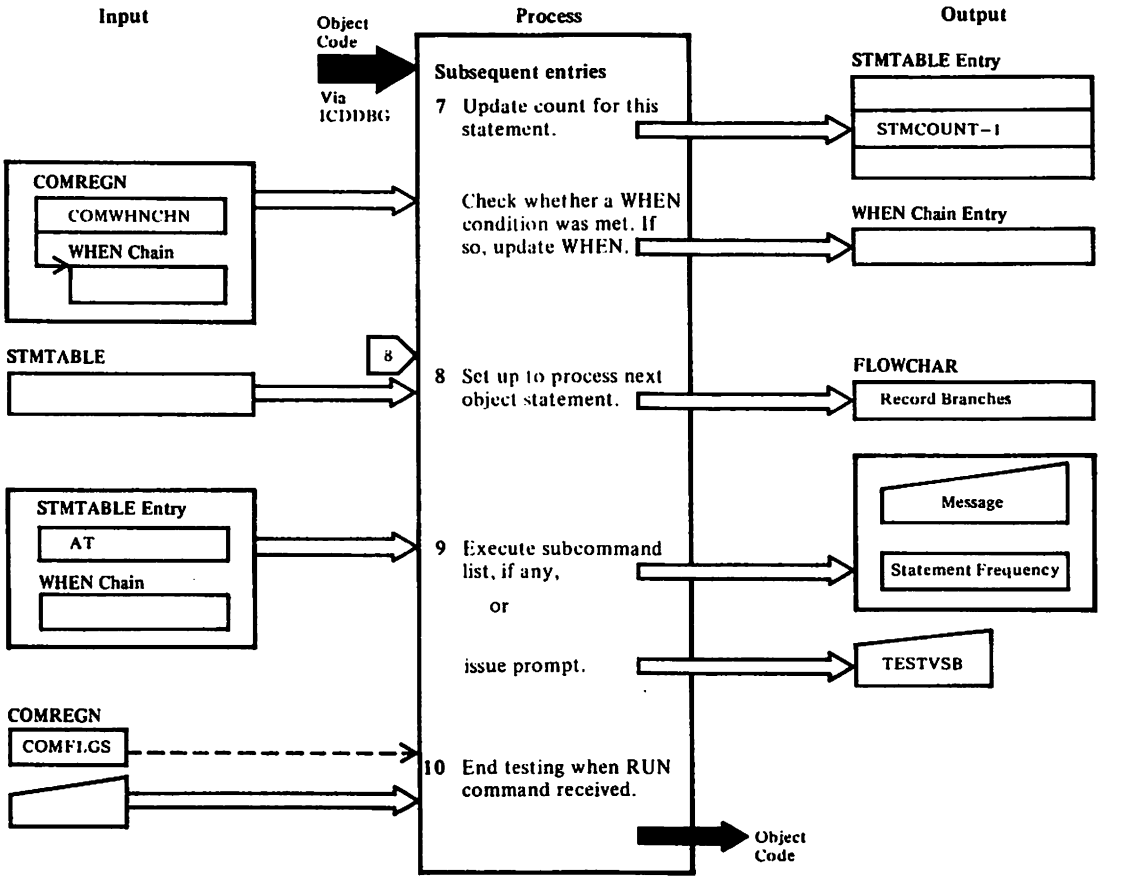


Diagram 9 (Part 3 of 5). Debug Processor

a WHEN condition specified entry to a program unit, the COMWHNCN pointer will be nonzero. Processing continues at step 8. [ICDWNTST]

ICDONITR

7 For any entry after the first, the STMCOUNT in the STMTABLE entry is incremented. ICDWNTST is called to check for and process any WHEN conditions which have been met. [ICDONITR]

8 For all entries, attention interrupts are disallowed, by setting the COMNINT flag to 1, while the current-statment boundary is updated to the next line number. If the statement is a branch, the FLOWCHAR table is updated. (FLOWCHAR maps the ten most recent branches in the program.)

The COMNEXT flag is checked. If it is on, the NEXT subcommand is in effect while interrupts are disallowed. In this case, the COMNXTRG flag is set to 1 and COMNEXT is turned off. Then, the COMATTEN flag is checked. If it is on, an ATTN occurred while interrupts were disallowed. This is noted by turning on COMATNTG. (At this point, interrupts are again allowed by setting COMNINT off.) The NEXT message is printed, and scanobey is called to process any subcommands the user enters. [ICDONITR,SCANOBEY]

9 The monitor checks whether COMRUNFL flag is on (a RUN was issued). The statement frequency count is incremented.

If required, a message for WHEN or NEXT is issued. If there is an AT for this statement entry in STMTABLE, the AT message is issued. Then ICDOBEY is called via scanobey to execute the subcommand list associated with the AT and to check for a statement range or additional text elements.

If there was no subcommand list, the TESTVSB prompt is issued. The statement frequency count is updated. Control is returned to the object code via ICDDBG.

The following list gives the routines that handle the scan phase and obey phase for each subcommand.

SUBCOMMAND	SCAN ROUTINE	OBEY ROUTINE
AT	ICDPSCL	ICDATTO
END	ICDPRSCN (ICD16SCN)	ICDOBEY (ICDENDO)
GO	ICDPRSCN (ICD0BSCN)	ICDGOGO
HALT	ICDPRSCN (ICD0ASCN)	ICDOBEY (ICDHALTO)
HELP	ICDPRSCN (ICD15SCN)	ICDHELPO
IF	ICDPRSCN	ICDIFO
LIST	ICDLSSCN	ICDLISTO
LISTBRKS	ICDPRSCN (ICD11SCN)	ICDLBKO
LISTFREQ	ICDPRSCN (ICD12SCN)	ICDLFQO
NEXT	ICDPRSCN (ICD05SCN)	ICDOBEY (ICDNEXTO)
OFF	ICDPRSCN (ICD08SCN)	ICDOFFO
OFFWHEN	ICDPRSCN (ICD26SCN)	ICDOFFWO

Diagram 9 (Part 4 of 5). Debug Processor

QUALIFY	ICDPRSCN (ICD04SCN)	ICDOBEY (ICDQUALO)
RUN	ICDPRSCN (ICD0CSCN)	ICDRUNO
SET	ICDSTSCN	ICDSETO
TRACE	ICDPRSCN (ICD27SCN)	ICDOBEY (ICDTRCO)
WHERE	ICDPRSCN (ICD13SCN)	ICDWHRO
WHEN	ICDPRSCN	ICDWHENO

Diagram 9 (Part 5 of 5). Debug Processor

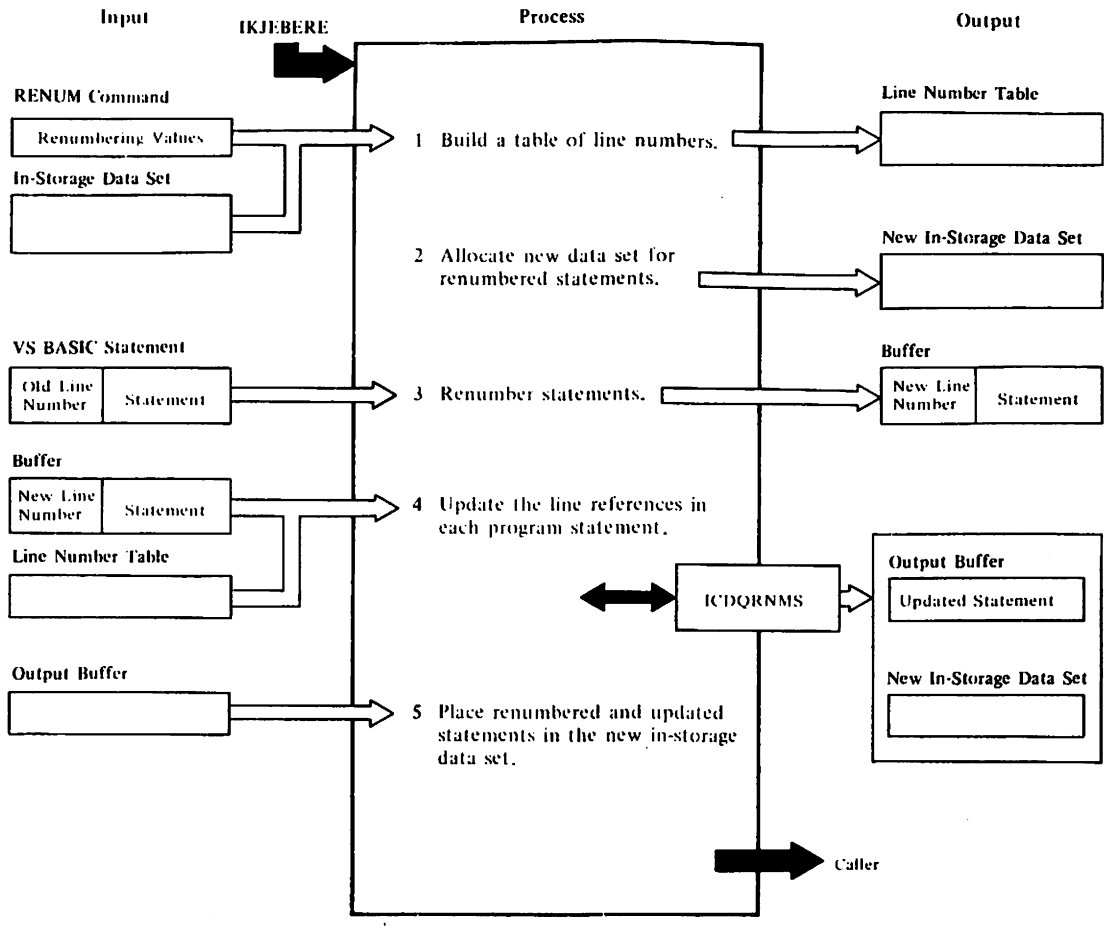


Diagram 10 (Part 1 of 2). Renumbering Facility -- TSO, CMS

ICDQRNME

1 The VS BASIC RENUMBER routine is passed the renumbering values, the in-core data set, its size and attributes. To perform the renumber operation two passes of the in-core data set are required. The first pass builds a number table that contains the existing line numbers and each corresponding new line number. [ICDQRNME]

2 The second pass allocates a new in-core data set that will contain the renumbered source. [ICDQRNME]

3 Each line is copied to a buffer with the new line number replacing the old one. [ICDQRNME]

4 The buffer is passed to a scanning routine (ICDQRNMS) to check for statement number references which are replaced by the corresponding new line numbers from the number table.

ICDQRNMS is passed an input buffer containing the source statement, a work area and an output buffer. Blanks are squeezed from the input buffer to allow scanning. Then the squeezed source statement, now in work buffer 1, is checked for statement number references and when found they are replaced by the new line numbers in the number table. The updated and squeezed source, now in work buffer 2, is then expanded into the output buffer by comparing the original record with the squeezed record to insert blanks where required. [ICDQRNME,ICDQRNMS]

5 The updated line then is placed in the new in-storage data set. On successful completion the in-storage data set size and location are updated to point to the new in-storage data set. On exit to IKJEBERE the number table and unused in-core data set are freed. [ICDQRNME]

Diagram 10 (Part 2 of 2). Renumbering Facility -- TSO, CMS

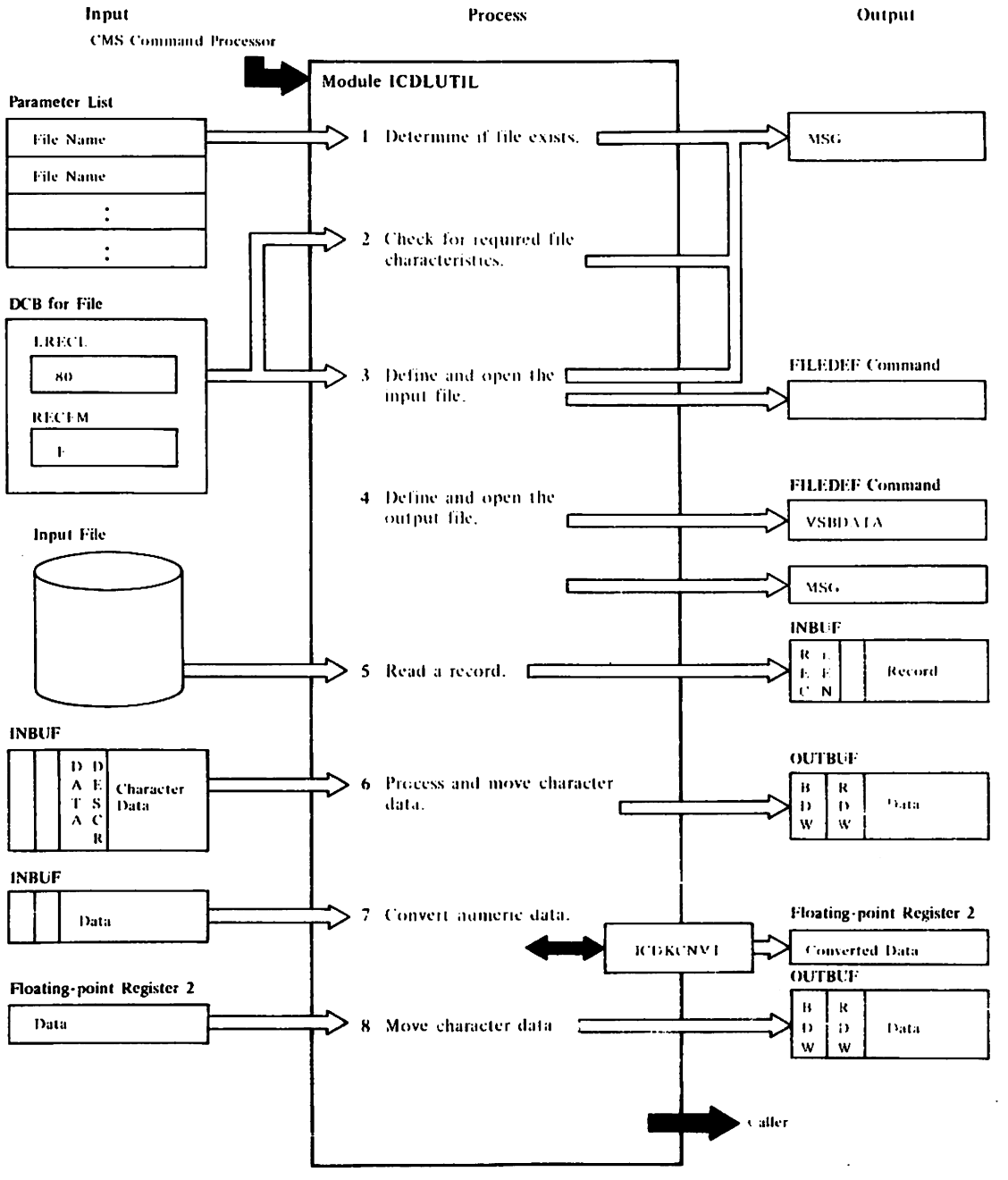


Diagram 11. File Conversion Utility -- CMS

This section of the manual lists in tabular form the entry points of the VS BASIC Processor modules with flow of control to and from each entry point.

- Table 1 lists the executor entry points
- Table 2 lists the compiler entry points
- Table 3 lists the library entry points
- Table 4 lists the debug entry points
- Table 5 lists the CMS conversion utility entry points
- Table 6 lists the renumbering facility entry points.

Table 1. VS BASIC Executor Module Entry Points (Part 1 of 4)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDQEXEC	ICDxEXEC (For TSO x is Q, for OS/VS x is Y, for CMS x is W, for DOS/VS x is Z, and for VSPC x is P)	System	Initializes the VS BASIC processor for a compilation or an execution		ICDJCMPA
STAEEXIT SPIEEXIT		SVC7/8	Process program checks		
STAXEXIT			Processes attention interrupt condition, sets flags for run-time routines before returning to point of interruption		System
SVCRET		SVC1, SVC2, SVC3, SVC4, SVC5, SVC6, SVC12, SVC13, SVC18, SVC21, or SVC22	Returns control to compiler language processing routines after a request for system service (SERV) has been processed		ICDJRUNA, ICDKCLOS, ICDKERRR, ICDKERRT, ICDKRUNX, ICDKERRS, ICDKCHN, ICDKETF2, ICDKOPN1, ICDKORGE, ICDKRSET ICDKTOUT, or ICDKCPUT
SVC0		STAXEXIT or SVC23, ICDKERRT or ICDKRUNX	Handles normal end of processing	TOUTPUT and CLEANUP	System

Table 1. VS BASIC Executor Module Entry Points (Part 2 of 4)

Entry Points	Module Name	Called By	Function	Calls	Exits To
SVC1		ICDKTOUT	Processes requests for terminal and unit record output	TOUTPUT	SVCRET
SVC2		ICDJERR ICDKPRNT ICDKINPT	Processes requests for terminal and unit record input	TINPUT and TOUTPUT	SVCRET
SVC3		SVC10, ICDKPUT, ICDKRSET	Processes requests for stream file output		SVCRET
SVC4		ICDKRSET ICDKGET	Processes requests for stream file input		SVCRET
SVC5		ICDJRUNA	Releases storage areas		SVCRET
SVC6		ICDKORGE	Acquires storage areas		SVCRET
SVC7		ICDKINTP	Processes returns from arithmetic interrupts for which a user exit exists		Caller
SVC8		ICDKINTP	Processes returns from arithmetic interrupts for which a user exit does not exist		SPIEEXIT
SVC9		ICDKFSCN	Encodes filename for VSPC		SVCRET
SVC10		ICDKCLOS, ICDKERRR, ICDKERRS, ICDKERRT, ICDKRUNX, ICDKCHN	Processes requests for stream file output and close	SVC3 and SVC22	SVCRET
SVC11		ICDJRUNA	Processes normal end of compilation and begins execution of the corresponding object program		Object Code or SVC0
SVC12		ICDKGSUB	Provides time and data information		
SVC13		Object Code	Processes PAUSE requests		SVCRET or SVC2
SVC14		ICDKINPT	Processes requests for retry of terminal input		
SVC16		ICDKCHN	Processes CHAIN statement requests to chain to a new program		
SVC18		ICDKCPU	Determines the amount of CPU time that has been used in the current terminal session		SVCRET
SVC21		ICDKOPN1	Processes requests to open stream files		SVCRET
SVC22		SVC10, ICDKCLOS, ICDKERRR, ICDKERRS, ICDKERRT, ICDKRUNX, ICDKCHN	Processes requests to close stream files		SVCRET

Table 1. VS BASIC Executor Modules Entry Points (Part 3 of 4)

Entry Points	Module Name	Called By	Function	Calls	Exits To
SVC23		ICDKERRS	Processes abnormal termination conditions	TOUTPUT	SVC0
SVC24		ICDKVDEL, ICDKVRD, ICDKVRRD, ICDKVRST, ICDKRWR, ICDKVWRT, ICDKVRRD ICDKINPT ICDKTOUT	Processes requests to process record-oriented files INPUT FROM PRINT TO		Caller
SVC25		ICDKVOPN, ICDKVTIO	Processes requests to open record-oriented files		Caller
SVC26		ICDKVCLS ICDKVEND	Processes requests to close record-oriented files		Caller
TINPUT		SVC2 and SVC14	Reads data from a terminal or unit record device		Caller
TOUTPUT		STAEXIT, SVC0, SVC1, SVC2, SVC23	Write data to a terminal or unit record device		
ICDPCLS	ICDPCLS	ICDPEXEC, SVC26	VSPC (OS AND DOS): Closes VSAM or VSPC files		Caller
ICDPOPN	ICDPOPN	ICDPEXEC, SVC25	VSPC (OS AND DOS): Opens VSAM or VSPC files		Caller
ICDPPENT	ICDPPENT	ICDPEXEC, SVC24	VSPC (OS and DOS): Does I/O for VSPC files		Caller
ICDPVENT	ICDPVENT	ICDPEXEC, SVC24	VSPC (OS and DOS): Does I/O for VSAM files		Caller
ICDQVCLS	ICDQVCLS	SVC26	Processes a VSAM CLOSE request		Caller
ICDQVDEL	ICDQVDEL	ICDQVENT	Processes a VSAM ERASE request		Caller or ICDQVERR
ICDQVENT	ICDQVENT	SVC24	Processes VSAM I/O requests	ICDQVDEL ICDQVGET ICDQVPUT ICDQVPNT	Caller
ICDQVERR	ICDQVERR	ICDQVDEL ICDQVGET ICDQVPNT ICDQVPUT	Processes VSAM logical and physical I/O errors and the end of file condition		Caller
ICDQVGET	ICDQVGET	ICDQVENT	Processes a VSAM READ request		Caller or ICDQVERR
ICDQVOPN	ICDQVOPN	SVC25	Processes a VSAM OPEN request		Caller
ICDQVPNT	ICDQVPNT	ICDQVENT	Processes a VSAM RESET request		Caller or ICDQVERR

Table 1. VS BASIC Executor Modules Entry Points (Part 4 of 4)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDQVPUT	ICDQVPUT	ICDQVENT	Processes a VSAM PUT request		Caller or ICDQVERR
ICDQWCLS	ICDQWCLS	SVC26	Processes a CMS VSAM CLOSE request		Caller
ICDQWDEL	ICDQWDEL	ICDQWENT	Processes a CMS VSAM ERASE request		Caller or ICDQWERR
ICDQWENT	ICDQWENT	SVC24	Processes CMS VSAM I/O requests	ICDQWDEL ICDQWGET ICDQWPNT ICDQWPUT	Caller
ICDQWERR	ICDQWERR	ICDQWDEL ICDQWGET ICDQWPNT ICDQWPUT	Processes CMS VSAM logical and physical I/O errors and the end of file condition		Caller
ICDQWGET	ICDQWGET	ICDQWENT	Processes CMS VSAM GET requests		Caller or ICDQWERR
ICDQWOPN	ICDQWOPN	SVC25	Processes CMS VSAM OPEN requests		Caller
ICDQWPNT	ICDQWPNT	ICDQWENT	Processes CMS VSAM RESET requests		Caller or ICDQWERR
ICDQWPUT	ICDQWPUT	ICDQWENT	Processes CMS VSAM PUT requests		Caller or ICDQWERR
ICDQZCLS	ICDQZCLS	SVC26	Processes a DOS/VS VSAM CLOSE request		Caller
ICDQZDEL	ICDQZDEL	ICDQZENT	Processes a DOS/VS VSAM ERASE request		Caller or ICDQZERR
ICDQZENT	ICDQZENT	SVC24	Processes DOS/VS VSAM I/O requests	ICDQZDEL ICDQZGET ICDQZPNT ICDQZPUT	Caller
ICDQZERR	ICDQZERR	ICDQZDEL ICDQZGET ICDQZPNT ICDQZPUT	Processes DOS/VS VSAM logical and physical I/O errors and the end of file condition		Caller
ICDQZGET	ICDQZGET	ICDQZENT	Processes DOS/VS VSAM GET requests		Caller or ICDQZERR
ICDQZOPN	ICDQZOPN	SVC25	Processes DOS/VS VSAM OPEN requests		Caller
ICDQZPNT	ICDQZPNT	ICDQZENT	Processes DOS/VS VSAM RESET requests		Caller or ICDQZERR
ICDQZPUT	ICDQZPUT	ICDQZENT	Processes DOS/VS VSAM PUT requests		Caller or ICDQZERR

Table 2. VS BASIC Compiler Module Entry Points (Part 1 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJAADJ	ICDJNUC1	ICDJALOC, ICDJDIM	Obtains storage for arrays and assigns displacements		Caller, or ICDJERRP
ICDJALOC	ICDJNUC1	ICDJMATD, ICDJFMLA	Creates array displacements and pointer entries	ICDJAADJ, ICDJVAL1	Caller
ICDJCDEF	ICDJDEFR	ICDJCEND	Controls the deferred compilation of the DATA, EXIT, FORM, and Image statements		ICDJCEND, ICDJERRP
ICDJCEND	ICDJNUCL	ICDJCLOS, ICDJGET, ICDJINPT, ICDJOPEN, ICDJPAUS, ICDJPRNT, ICDJPUT, ICDJRSET, ICDJRSTO, ICDJREAD, ICDJMATV, ICDJDIMG, ICDJFOR, ICDJLET, ICDJCMPA, ICDJIMAG, ICDJON, ICDJCDEF, ICDJFORM, ICDJEXIT, ICDJDATA, ICDJEND, ICDJCHN, ICDJERRP, ICDJNEXT, ICDJGOSB, ICDJGOTO, ICDJFNE1, ICDJFNE2, ICDJDEF1, ICDJDEF2, ICDJRETN, ICDJSTOP, ICDJUSE, ICDJDIM, ICDJVREC	Common exit point for all normal exits from the statement processing routines	ICDJDEF2, ICDJDFRM ICDJDIMG ICDJFNE1 ICDJERRP, ICDJERRS, ICDJERRT, ICDJLET, ICDJVAL, ICDJVDEL, ICDJVRD, ICDJVRRD, ICDJVRWR, ICDJVWRT,	ICDJCDEF, ICDJCTL, ICDJERRP, ICDJERRS, or ICDJERRT
ICDJCHN	ICDJVERB	ICDJCTL	Processes the CHAIN statement	ICDJFMLA	ICDJCEND or ICDJERRP
ICDJCLOS	ICDJIOVB	ICDJCTL	Processes CLOSE statements; emits code that links to the run-time routine ICDKCLOS	ICDJFMLA	ICDJCEND or ICDJERRS
ICDJCMPA	ICDJCMPA	ICDxEXEC	Suballocates user area and initializes it for a compilation	SVC8	ICDJCTL ICDJERRS or ICDJERRT

Table 2. VS BASIC Compiler Module Entry Points (Part 2 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJCNVT	ICDJNUCL	ICDJPRNT, ICDJLINE, ICDJSCN1, ICDJFORM, ICDJEXIT, ICDJDATA, ICDJDIM	Converts numeric EBCDIC fields to binary		Caller or ICDJERRP
ICDJCONF	ICDJNUCL	ICDJSCN1	Converts EBCDIC numbers to floating-point binary		Caller or ICDJERRP
ICDJCTL	ICDJNUCL	ICDJCEND ICDJCMPA	Controls the in-line compilation of all VS BASIC statements except DATA, EXIT, FORM, and Image. It processes the line number, determines the statement type, and branches to the appropriate statement processing routine.	ICDJLINE, ICDJVAL1	All Statement Processing Routines (except ICDJDATA ICDJEXIT ICDJFORM ICDJIMAG), ICDJERRP, or ICDJERRT
ICDJDATA	ICDJDEFR	ICDJCDEF	Processes the DATA statement; stores values in a data block, processes literal data and replication factor.	ICDJCNVT	ICDJCEND, ICDJRUNA, ICDJERRP, or ICDJERRT
ICDJDEF1 ICDJDEF2	ICDJUSFN	ICDJCTL	Processes user function definitions	ICDJSCN1, ICDJRETV ICDJFNE2,	ICDJCEND or ICDJERRP
ICDJDIM	ICDJVERB	ICDJCTL	Processes the DIM statement	ICDJSCN at ICDJSCN1, ICDJCNVT, ICDJVAL2	ICDJCEND or ICDJERRP
ICDJDIMG	ICDJNUCL	ICDJCEND	Processes the Image statement		ICDJCEND
ICDJEND	ICDJDEFR	ICDJCTL	Processes the END statement		ICDJCDEF via ICDJCEND or ICDJERRP
ICDJERRN	ICDJERR	All Statement Processing Routines	Prints error messages for errors that terminate compilation		ICDJERRT
ICDJERRP	ICDJERR		Prints line numbers and syntax error messages for errors that do not terminate compilation		ICDJCEND

Table 2. VS BASIC Compiler Module Entry Points (Part 3 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJERRS	ICDJERR		Returns to the executor following processing of errors that result in abnormal termination of compilation		SVC23
ICDJERRT	ICDJERR		Prints line numbers and error messages for errors that terminate compilation		SVC0
ICDJEXIT	ICDJDEFR	ICDJCDEF	Processes the EXIT statement	ICDJCNVT	ICDJCEND or ICDJERRP
ICDJFBN1	ICDJFUTS	ICDJFMLA	Processes binary operator expressions where both operands are in floating-point registers		ICDJFRM5
ICDJFBN2	ICDJFUTS	ICDJFMLA	Processes binary operator expressions where the first operand is in storage and the second is in a floating-point register		ICDJFRM5
ICDJFCAT	ICDJFUTS	ICDJFMLA	Processes character expressions involving the concatenation operator		ICDJFRM3 or ICDJERRP
ICDJFEXP	ICDJFUTS	ICDJFMLA	Processes exponentiation expressions		ICDJFRM5
ICDJFGEN	ICDJFUTS	ICDJFMLA	Emits in-line code for intrinsic functions		ICDJFRM2
ICDJFMLA	ICDJNUC1	ICDJCLOS, ICDJGET, ICDJFOR, ICDJGOSB, ICDJIF, ICDJCHN, ICDJPRNT, ICDJPUT, ICDJMATV, ICDJRETV, ICDJRDIM, ICDJRSET, ICDJSCN1, ICDJSCN2, ICDJFOR, ICDJLET,	Processes single arithmetic and character expressions and lists of arithmetic and character expressions	ICDJSCN at ICDJSCN2, ICDJFCAT, ICDJALOC, ICDJFEXP, ICDJFBN1, ICDJFBN2, ICDJFGEN, ICDJFUNY,	Caller or ICDJERRP
ICDJFNE1	ICDJUSFN	ICDJCTL	Processes the FNEND statement for multi-line user function definitions		ICDJCEND or ICDJERRP
ICDJFNE2	ICDJUSFN	ICDJDEF	Processes single line user function definitions		ICDJCEND or ICDJERRP
ICDJFOR	ICDJNUC2	ICDJCTL	Processes the FOR statement	ICDJFMLA, ICDJVAL1, or ICDJSCN1	ICDJCEND or ICDJERRP

Table 2. VS BASIC Compiler Module Entry Points (Part 4 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJFORM	ICDJDEFR	ICDJCDEF	Processes the FORM statement	ICDJSCN1, ICDJCNVT	ICDJCEND or ICDJERRP
ICDJFRM2	ICDJNUC1	ICDJFGEN	Return point from code generation		
ICDJFRM3	ICDJNUC1	ICDJFCAT, ICDJFUNY	Return point from code generation		
ICDJFRM5	ICDJNUC1	ICDJFBN1, ICDJFBN2, ICDJFEXP	Return point from code generation		
ICDJFUNY	ICDJFUTS	ICDJFMLA	Processes unary operator expressions		ICDJFRM3
ICDJGET	ICDJIOVB	ICDJCTL ICDJMATV	Processes the GET statement; emits code to link to the run-time routine ICDKGET	ICDJSCN1, ICDJFMLA	ICDJCEND or ICDJERRS
ICDJGOSB	ICDJNUC4	ICDJCTL	Processes the GOSUB statement	ICDJLINE, ICDJFMLA	ICDJCEND or ICDJERRS
ICDJGOTO	ICDJNUC4	ICDJCTL	Processes the GOTO statement	ICDJLINE	ICDJCEND or ICDJERRS
ICDJIF	ICDJNUC5	ICDJCTL	Processes the IF statement	ICDJFMLA, ICDJLINE	ICDJCEND or ICDJERRP
ICDJIF1	ICDJNUC5		Return point after processing the THEN clause	ICDJLINE	
ICDJIF2	ICDJNUC5		Return point after processing the ELSE clause		
ICDJINFO	ICDINFO		Contains the compiler Information Table		
ICDJIMAG	ICDJDEFR	ICDJCDEF	Processes the Image statement		ICDJCEND or ICDJERRP
ICDJINPT	ICDJIOVB	ICDJCTL	Processes the INPUT statement; emits code to link to the run-time routine ICDKINPT	ICDJSCN1	ICDJCEND or ICDJERRS
ICDJLET	ICDJNUC2	ICDJCTL	Processes assignment statements	ICDJFMLA, ICDJSCN1	ICDJCEND or ICDJERRP
ICDJLINE	ICDJNUCL	ICDJGOSB, ICDJGOTO, ICDJIF1, ICDJIF2	Scans the source code for valid line numbers	ICDJCNVT	Caller or ICDJERRS

Table 2. VS BASIC Compiler Module Entry Points (Part 5 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJMATD	ICDJMATV	ICDJMATV	Allocates arrays defined in MAT statements	ICDJRDIM, ICDJALOC	Caller or ICDJERRS
ICDJMATV	ICDJMATV	ICDJCTL	Determines the type of matrix operation required; emits code to link to the run-time routines for the specified operation	ICDJFMLA, ICDJRDIM, ICDJMATD, ICDJVRD, ICDJREAD, ICDJPUT, ICDJGET	ICDJCEND or ICDJERRS
ICDJNEXT	ICDJNUC3	ICDJCTL	Processes the NEXT statement	ICDJSCN1	ICDJCEND, ICDJERRP, or ICDJERRT
ICDJON	ICDJVERB	ICDJCTL	Processes the ON statement		ICDJCEND
ICDJOPEN	ICDJIOVB	ICDJCTL	Processes the OPEN statements	ICDJFMLA	ICDJCEND or ICDJERR
ICDJPAUS	ICDJIOVB	ICDJCTL	Processes the PAUSE statement		ICDJCEND or ICDJERR
ICDJPRNT	ICDJIOVB	ICDJCTL	Processes the PRINT statement; emits code to link to the run-time routine ICDKPRNT	ICDJCNVT, ICDJFMLA	ICDJCEND or ICDJERRS
ICDJPUT	ICDJIOVB	ICDJCTL, ICDJMATV	Processes the PUT statement; emits code to link to the run-time routine ICDKPUT	ICDJFMLA, ICDJFEXT	ICDJCEND or ICDJERRS
ICDJRDIM	ICDJMATV	ICDJMATD ICDJSCN1	Processes redimensioning of arrays	ICDJFMLA	Caller
ICDJREAD	ICDJIOVB	ICDJCTL ICDJMATV	Processes the READ statement	ICDJSCN1	ICDJCEND or ICDJERRS
ICDJRETN	ICDJUSFN	ICDJCTL	Processes GOSUB and multiline user function RETURN statements	ICDJRETV	ICDJCEND or ICDJERRP
ICDJRETV	ICDJUSFN	ICDJRETN ICDJDEF	Produces code to return a value for user functions	ICDJFMLA	Caller to ICDJERRP
ICDJRSET	ICDJIOVB	ICDJCTL	Processes the RESET statement; emits code to link to the run-time routine ICDKRSET	ICDJFMLA	ICDJCEND or ICDJERRS
ICDJRSTO	ICDJIOVB	ICDJCTL	Processes the RESTORE statement		ICDJCEND
ICDJRUNA	ICDJRUNA	ICDJDATA	Prepares the user area for storing or executing the object program produced by a compilation	SVC5 or SVC10	SVC11 or ICDJERRP

Table 2. VS BASIC Compiler Module Entry Points (Part 6 of 6)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDJSCN1	ICDJNUC1	ICDJGET, ICDJINPT, ICDJREAD, ICDJFOR, ICDJLET ICDJFORM, ICDJNEXT, ICDJDEF1, ICDJDEF2, ICDJDIM	Scans the source statements for valid identifiers (simple variables, literals, internal constants, user functions, array elements, and arrays)	ICDJCONF, ICDJCNVT, ICDJFMLA, ICDJVAL	Caller or ICDJERRT
ICDJSCN2	ICDJNUC1	ICDJFMLA	Identifies operands in expressions		
ICDJSTOP	ICDJVERB	ICDJCTL	Processes the STOP statement		ICDJCEND
ICDJUSE	ICDJVERB	ICDJCTL	Processes the USE statement; emits data used by the run-time routine ICDKORGE		ICDJCEND or ICDJERRP
ICDJVAL1	ICDJNUC1	ICDJSCN1	Allocates storage for arithmetic variables		Caller or ICDJERRT
ICDJVAL2	ICDJNUC1		Allocates storage for alphameric variables on a word boundary		
ICDJVAL3	ICDJNUC1	ICDJSCN1	Allocates storage for alphameric variables on other than a word boundary		
ICDJVAL4	ICDJNUC1		Determines if sufficient storage is available to allocate to a variable; allocates storage, if available, or indicates an error for insufficient storage		
ICDJVAL5	ICDJNUC1	ICDJSCN1 ICDJDATA	Allocates storage for character constant and initializes with literal value		Caller
ICDJVDEL	ICDJVREC	ICDJCTL	Processes the record I/O ERASE statement	ICDJFMLA, ICDJDEXT, ICDJCNVT ICDJSCN1	ICDJCEND or ICDJERRP
ICDJVRD	ICDJVREC		Processes the record I/O READ statement		
ICDJVRRD	ICDJVREC		Processes the record I/O REREAD statement		
ICDJVRWR	ICDJVREC		Processes the record I/O REWRITE statement		
ICDJVWRT	ICDJVREC		Processes the record I/O WRITE statement		

Table 3. VS BASIC Library Module Entry Points (Part 1 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKACS	ICDKSSUB	Object Code	Evaluates the arc cosine function (ACS)	ICDKSQR	Caller
ICDKASN	ICDKSSUB	Object Code	Evaluates the arc sine function (ASN)		Caller
ICDKATN	ICDKSSUB	Object Code	Evaluates the arc tangent function (ATN)		Caller
ICDKBFTB	ICDKBFTB		Table of branch addresses for the run-time routines		
ICDKCHN	ICDKERR		Processes a request to chain the execution of a new program to the end of the current one		SVC16
ICDKCHR	ICDKGSUB	Object Code	Evaluates the numeric to character function (CHR)	ICDKCNVT	Caller
ICDKCLK	ICDKGSUB	Object Code	Evaluates the time of day function (CLK)	SVC12	Caller
ICDKCLOS	ICDKIOVB	Object Code	Calls the executor to write and close a stream file if the output buffer contains data or to close the file if the buffer is empty	ICDKFSCN, SVC10, or SVC22	Caller, ICDKERRS, or ICDKERRT
ICDKCNVT	ICDKCNVT	ICDKPRNT, ICDKPUT, ICDKVRWR, or ICDKVWRT	Converts floating-point numbers according to the format specified by an unformatted print, a PIC specification, a FORM statement, or an Image statement	ICDKPLIN ICDKTOUT	Caller
ICDKCOS	ICDKSSUB	Object Code	Evaluates the cosine function (COS)		Caller
ICDKCOT	ICDKSSUB	Object Code	Evaluates the cotangent function (COT)		Caller
ICDKCPU	ICDKGSUB	Object Code	Evaluates the program execution time function (CPU)	SVC18	Caller
ICDKCSC	ICDKSSUB	Object Code	Evaluates the cosecant function (CSC)	ICDKCOS	Caller
ICDKDABS		Object Code	Evaluates the absolute function (ABS)		Caller
ICDKDACS	ICDKDSUB	Object Code	Evaluates the double precision arc cosine function (ACS)	ICDKSQR	Caller
ICDKDAIN		Object Code	Evaluates the integer function (INT)		Caller

Table 3. VS BASIC Library Module Entry Points (Part 2 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKDASN	ICDKDSUB	Object Code	Evaluates the double precision arc sine function (ASN)		Caller
ICDKDAT	ICDKGSUB	Object Code	Evaluates the date function (DAT)		Caller
ICDKDATN	ICDKDSUB	Object Code	Evaluates the double precision arc tangent function (ATN)		Caller
ICDKDBPR	ICDKPRNT	ICDLISTO	Formats and prints arrays for the debug processor		Caller
ICDKDCOS	ICDKDSUB	Object Code	Evaluates the double precision cosine function (COS)		Caller
ICDKDCOT	ICDKDSUB	Object Code	Evaluates the double precision cotangent function (COT)		Caller
ICDKDCSC	ICDKDSUB	Object Code	Evaluates the double precision cosecant function (CSC)	ICDKDCOS	Caller
ICDKDET	ICDKMINV	Object Code	Evaluates the matrix determinant function (DET)	SVC5 or SVC6	Caller or ICDKERRT
ICDKDEXP	ICDKDSUB	Object Code	Evaluates the double precision exponent function (EXP)		Caller
ICDKDHCS	ICDKDSUB	Object Code	Evaluates the double precision hyperbolic cosine function (HCS)	ICDKDEXP	Caller
ICDKDHSN	ICDKDSUB	Object Code	Evaluates the double precision hyperbolic sine function (HSN)		Caller
ICDKDHTN	ICDKDSUB	Object Code	Evaluates the double precision hyperbolic tangent function (HTN)	ICDKDEXP	Caller
ICDKDLGT	ICDKDSUB	Object Code	Evaluates the double precision log base 10 function (LGT)		Caller
ICDKDLOG	ICDKDSUB	Object Code	Evaluates the double precision log base 2 function (LOG)		Caller
ICDKDLTW	ICDKDSUB	Object Code	Evaluates the double precision log base e function (LTW)		Caller
ICDKDMAX	ICDKDSUB	Object Code	Evaluates the double precision maximum value function (MAX)		Caller
ICDKDMIN	ICDKDSUB	Object Code	Evaluates the double precision minimum value function (MIN)		Caller
ICDKDOT	ICDKMAT	Object Code	Evaluates the matrix dot product function (DOT)		Caller or ICDKERRT
ICDKDPWR	ICDKDSUB	Object Code	Evaluates double precision internal exponentiation operation	ICDKDLOG, ICDKDEXP	Caller

Table 3. VS BASIC Library Module Entry Points (Part 3 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKDSEC	ICDKDSUB	Object Code	Evaluates the double precision secant function (SEC)		Caller
ICDKDSIN	ICDKDSUB	Object Code	Evaluates the double precision sine function (SIN)		Caller
ICDKDSQR	ICDKDSUB	Object Code	Evaluates the double precision square root function (SQR)		Caller
ICDKDTAN	ICDKDSUB	Object Code	Evaluates the double precision tangent function (TAN)		Caller
ICDKERRR	ICDKERR	ICDKINPT ICDKINTP	Writes an error message and returns to the caller	SVC10 SVC22 SVC1	Caller
ICDKERRS	ICDKERR	ICDKCLOS, ICDKFSCN, ICDKINPT, ICDKINTP, ICDKOPEN ICDKPUT ICDKREAD	Writes a system error message and terminates execution of the object program		SVC23
ICDKERRT	ICDKERR	ICDKCLOS ICDKCNVT ICDKETOF ICDKETF2 ICDKGET ICDKKLN ICDKKPS ICDKMINV ICDKOPN1 ICDKPRNT ICDKPUT ICDKREAD ICDKRLN ICDKRST	Writes an error message and terminates execution of the object program or passes control to user-specified line number		SVC0
ICDKETF2	ICDKETOF	ICDKGET ICDKINPT ICDTSCN	Moves character data to the user target area	SVC4	Caller or ICDKERRT
ICDKETOF	ICDKETOF	ICDKGET ICDKINPT ICDTSCN	Converts and moves numeric data into a user target area	SVC4	Caller or ICDKERRT
ICDKEXP	ICDKSSUB	Object Code	Evaluates the exponent function (EXP)		Caller
ICDKFSCN	ICDKIOVB	ICDKCLOS, ICDKGET, ICDKOPEN, ICDKPUT, ICDKRLN, ICDKRSET, ICDKVOPN	Checks if a file referenced in an I/O statement is open and if not is there an available place in which to open it		Caller or ICDKERRT

Table 3. VS BASIC Library Module Entry Points (Part 4 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKGET	ICDKIOVB	Object Code	Processes stream file input requests	ICDKFSCN, ICDKOPN1, ICDKETF2, ICDKETOF, SVC4	Caller or ICDKERRT
ICDKHCS	ICDKSSUB	Object Code	Evaluates the hyperbolic cosine function (HCS)		Caller
ICDKHSN	ICDKSSUB	Object Code	Evaluates the hyperbolic sine function (HSN)		Caller
ICDKHTN	ICDKSSUB	Object Code	Evaluates the hyperbolic tangent function (HTN)		Caller
ICDKIDX	ICDKGSUB	Object Code	Evaluates the string position function (IDX)		Caller
ICDKINPT	ICDKINPT	Object Code	Processes unit record input requests	ICDKETF2, ICDKETOF, ICDKTOUT, SVC2, SVC14, or SVC24	Caller, ICDKERRR, or ICDKERRS
ICDKINTP	ICDKINTP		Processes run-time arithmetic interrupts	ICDKERRR	SVC8 or ICDKERRS
ICDKJDY	ICDKGSUB	Object Code	Evaluates the Julian date function (JDY)		Caller
ICDKKLN	ICDKKLN	Object Code	Evaluates the key length function (KLN)		Caller or ICDKERRT
ICDKKPS	ICDKKPS	Object Code	Evaluates the key position function (KPS)	ICDKFSCN	Caller or ICDKERRT
ICDKLEN	ICDKGSUB	Object Code	Evaluates the string length function (LEN)		Caller
ICDKLGT	ICDKSSUB	Object Code	Evaluates the log base 10 function (LGT)		Caller
ICDKLOG	ICDKSSUB	Object Code	Evaluates the log base e function (LOG)		Caller
ICDKLTW	ICDKSSUB	Object Code	Evaluates the log base 2 function (LTW)		Caller
ICDKMADD	ICDKMAT	Object Code	Processes matrix addition		Caller or ICDKERRT
ICDKMASN	ICDKMAT	Object Code	Processes matrix assignment		Caller or ICDKERRT
ICDKMASR	ICDKMAT	Object Code	Processes matrix ascending sort		Caller or ICDKERRT
ICDKMAX	ICDKSSUB	Object Code	Evaluates the maximum value function (MAX)		Caller

Table 3. VS BASIC Library Module Entry Points (Part 5 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKMDSR	ICDKMAT	Object Code	Processes matrix descending sort		Caller or ICDKERRT
ICDKMIDN	ICDKMAT	Object Code	Evaluates the matrix identity function (IDN)		Caller or ICDKERRT
ICDKMIN	ICDKSSUB	Object Code	Evaluates the minimum value function (MIN)		Caller
ICDKMINV	ICDKMINV	Object Code	Processes matrix inversion	SVC5 or SVC6	Caller or ICDKERRT
ICDKMMUL	ICDKMAT	Object Code	Processes matrix multiplication		Caller or ICDKERRT
ICDKMSCA	ICDKMAT	Object Code	Processes the assignment of a scalar to a matrix		Caller or ICDKERRT
ICDKMSUB	ICDKMAT	Object Code	Processes matrix subtraction		Caller or ICDKERRT
ICDKMTRN	ICDKMAT	Object Code	Processes matrix transposition		Caller or ICDKERRT
ICDKNUM	ICDKGSUB	Object Code	Evaluates the character string value function (NUM)	ICDKETOF	Caller
ICDKON	ICDKERR	Object Code	Activates/deactivates ON conditions		Caller
ICDKOPEN	ICDKIOVB	Object Code	Processes stream file open requests	ICDKOPN1	Caller or ICDKERRT
ICDKOPN1	ICDKOPN2	ICDKOPEN	Opens a stream file	SVC21	Caller or ICDKERRT
ICDKORGE	ICDKORGE	Object Code from PRBGN in PRG	Prepares a VS BASIC program for execution (sets the current line width, handles chaining, and allocates and initializes arrays)	SVC6	Object Code via BGEX in OBJAREA
ICDKPLIN	ICDKPLIN	ICDKCNVT, ICDKPRNT	Moves character strings to the print buffer	ICDKTOUT	Caller
ICDKPRD	ICDKMAT	Object Code	Evaluates the matrix product function (PRD)		Caller or ICDKERRT
ICDKPRNT	ICDKPRNT	Object Code	Controls the conversion and formatting of data specified in PRINT, PRINT USING, MAT PRINT, and PAUSE statements	ICDKCNVT, ICDKPLIN, ICDKTOUT SVC13 SVC1	Caller, ICDKERRS, or ICDKERRT
ICDKPUT	ICDKIOVB	Object Code	Processes stream file output requests	ICDKFSCM ICDKOPN1 ICDKETOF SVC3	Caller
ICDKPWR	ICDKSSUB	Object Code	Processes internal exponentiation operations	ICDKEXP or ICDKLOG	Caller

Table 3. VS BASIC Library Module Entry Points (Part 6 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKRDM1	ICDKMAT	Object Code or ICDREDIM	Redimensions an array to one dimension		Caller or ICDKERRT
ICDKRDM2	ICDKMAT	Object Code or ICDREDIM	Redimensions an array to two dimensions		Caller or ICDKERRT
ICDKREAD	ICDKREAD	Object Code	Read values in a DATA statement into variables as specified in a READ statement		Caller, ICDKERRS, or ICDKERRT
ICDKRLN	ICDKRLN	Object Code	Evaluates the last record, length function (RLN)	ICDKFSCN	Caller or ICDKERRT
ICDKRND	ICDKSSUB	Object Code	Evaluates the random number function (RND)		Caller
ICDKRSET	ICDKIOVB	Object Code	Resets a stream file to the beginning or to the end and opens the file if necessary	ICDKFSCN, ICDKOPN1, SVC3, or SVC4	Caller or ICDKERRT
ICDKRUNX	ICDKERR	Object Code or ICDOBEY	Handles the normal termination of a program	ICDKTOUT, SVC10 or SVC22 ICDDBG4 ICDDBG5	SVC0
ICDKRUNY	ICDKERR	STAXEXIT	Handles the normal termination of a program after an attention interrupt		SVC0
ICDKSEC	ICDKSSUB	Object Code	Evaluates the secant function (SEC)	ICDKSIN	Caller
ICDKSIN	ICDKSSUB	Object Code	Evaluates the sine function (SIN)		Caller
ICDKSQR	ICDKSSUB	Object Code	Evaluates the square root function (SQR)		Caller
ICDKSTR	ICDKGSUB	Object Code	Evaluates the string position function (STR)		Caller
ICDKSUM	ICDKMAT	Object Code	Evaluates the matrix sum function (SUM)		Caller or ICDKERPT
ICDKTAN	ICDKSSUB	Object Code	Evaluates the tangent function (TAN)		Caller
ICDKTIM	ICDKGSUB	Object Code	Evaluates the time of day function (TIM)		Caller

Table 3. VS BASIC Library Module Entry Points (Part 7 of 7)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDKTIO	ICDKPRNT	Object Code	Handles switching of terminal input/output resulting from PRINT TO or INPUT FROM statements	ICDKTOUT ICDKVTIO	Caller
ICDKTOUT	ICDKTOUT	ICDKCNVT, ICDKERRS, ICDKERRT, ICDKINPT, ICDKPLIN, ICDKPRNT, or ICDKRUNX	Move the internal buffer to the terminal or unit record output buffer	SVC1 SVC24	Caller
ICDKVCLS	ICDKVIOR	Object Code	Closes a VSAM file	SVC26	Caller or SVC13
ICDKVDEL	ICDKVIOR	Object Code	Erases a VSAM file	SVC24	Caller
ICDKVEND	ICDKVIOR				
ICDKVOPN	ICDKVIOR	Object Code, ICDKVRD, ICDKVRST, ICDKVRWR, ICDKVRD	Locates and opens a VSAM file	ICDKFSCN, SVC25	Caller or SVC13
ICDKVRD	ICDKVIOR	Object Code	Reads a VSAM file	SVC24	Caller
ICDKVRRD	ICDKVIOR	Object Code	Rereads a VSAM file	SVC24	Caller
ICDKVRST	ICDKVIOR	Object Code	Resets a VSAM file	ICDKCNVT,	Caller
ICDKVRWR	ICDKVIOR	Object Code	Rewrites a VSAM file	IDCKCNVT, SVC24	Caller
ICDKVTIO	ICDKVIOR	ICDKTIO	Handles implicit OPEN of a 'terminal' file	ICDKFSCN, SVC25	Caller
ICDKVWRT	ICDKVIOR	Object Code	Writes a VSAM file	ICDKCNVT, SVC24	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 1 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDADRES	ICDADRES	ICDEVALU ICDLISTO ICDSETO ICDWHENO ICDWNTST	Resolves addresses for the debug routines	ICDFOSUB ICDMSSG	Caller
ICDATTN	ICDATTN	ICDONITR	Handles an attention interrupt entered during debugging. Output being produced by a debug subcommand is terminated and a subcommand is requested and an immediate subcommand is processed	ICDBLNKL ICDDSCAN ICDMODE ICDMSSG ICDWHRO	Caller
ICDATTO	ACDATTO	ICDOBEY	Executes the AT subcommand	ICDDECHN	Caller
ICDBLDTB	ICDBLDTB	ICDKORGE	Builds tables for use by the VS BASIC Debug Processor		Caller
ICDCCHN	ICDCHAIN	ICDTSCN ICDWHENO	Creates a text element for constants		Caller
ICDCDSCN	ICDCDSCN	ICDWHSCN	Scans the WHEN and IF conditions	ICDNSCAN	Caller
ICDCHAIN	ICDCHAIN	ICDIDCHK ICDISCAN ICDLSSCN ICDOFFWO ICDON02 ICDPRSCN ICDPSCL ICDSSCN ICDTSCN ICDTSTYP ICDWHENO ICDWNSCN ICDZERO ICD04SCN ICD05SCN ICD08SCN ICD0ASCN ICD0BSCN ICD0CSCN ICD11SCN ICD12SCN ICD13SCN ICD15SCN ICD16SCN ICD26SCN ICD27SCN	Creates and processes chained text elements		Caller
ICDCMTBL	ICDCMTBL		Tables of subcommand names and processing routines		
ICDCOMR	ICDCOMR		VS BASIC Debug communications region		
ICDDBG	ICDDBG	Object Code	Interface between the object code and the debug monitor routine (ICDONITR). This routine is called at each statement boundary to check for debugging information	ICDONITR ICDMSSG ICDDECHN ICDISCAN ICDPROMT	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 2 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDDBG2	ICDDBG	Object Code	Processes trace information on entering a user function	ICDON02	Caller
ICDDBG3	ICDDBG	Object Code	Processes trace information on exiting from a user function	ICDON03	Caller
ICDDBG4	ICDDBG	ICDKRUNX	Processes the normal termination of a user's program under debug	ICDON04	Caller
ICDDBG5	ICDDBG	ICDKRUNX	Processes an abnormal termination of a user's program under debug	ICDON05	Caller
ICDDECHN	ICDCHAIN	ICDATTO ICDCDSCN ICDISCAN ICDLSSCN ICDOBEY ICDOFFO ICDOFFWO ICDON03 ICDDECHN ICDRUNO ICDSCAN IDSSCN ICDSTSCN ICDWHENO ICDWNSCN ICDTSTYP	Removes text elements from a text chain		Caller
ICDDSCAN	ICDDSCAN	ICDATTN ICDSCAN ICDSSCAN	Scans a subcommand for validity and routines control to the appropriate processing routine	ICDMCMDS ICDSCMDS	Caller
ICDDTMSG	ICDMSSG	ICDLISTO	Prints character and numeric scalar variables for the LIST subcommand		Caller
ICDEVALU	ICDEVALU	ICDIFOB ICDWNTST	Evaluates the conditions defined in the IF and WHEN subcommands and determines whether the condition is true or false	ICDADRES	Caller
ICDFLOW	ICDFLOW	ICDWHRO	Evaluates the WHERE statement	ICDMSSG ICDSTCNV	Caller
ICDFOSUB	ICDFOSUB	ICDADRES ICDLISTO	Formats substring array elements	ICDKCNVT	Caller
ICDGOGO	ICDGOGO	ICDOBEY ICDRUNO	Executes the GO TO subcommand	ICDMSSG	Caller
ICDHELPO	ICDHELPO	ICDOBEY	Executes the HELP subcommand (TSO only), attaches the TSO HELP processor, and uses the TSO service routine (IKJDAIR) to process the HELP subcommand	IKJDAIR	Caller
ICDIDCHK	ICDIDCHK	ICDPMACS	Scans for a valid statement line number	ICDCHAIN ICDDECHN ICDMSSG ICDTSRCH	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 3 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDIFOB	ICDIFOB		Executes the IF subcommand	ICDEVALU	Caller
ICDIFSCN			Alias for ICDWNSCN		
ICDIINS1	ICDISCAN		Insertion segment that is passed to the message processing routine ICDMSSG		
ICDISCAN	ICDISCAN	ICDCDSCN ICDLSSCN ICDSTSCN ICDWNSCN	Scans VS BASIC items that appear in the SET, LIST, WHEN, and IF subcommands	ICDCHAIN ICDDECHN ICDMSSG ICDPROMT ICDTSCN	Caller
ICDLBKO	ICDLBKO	ICDOBEY	Executes the LISTBRKS subcommand	ICDMSSG ICDSTCNV	Caller
ICDLFQO	ICDLFQO	ICDOBEY	Executes the LISTFREQ subcommand	ICDMSSG ICDSTCNV	Caller
ICDLISTO	ICDLISTO	ICDOBEY	Executes the LIST subcommand	ICDADRES ICDDTMSG ICDFOSUB ICDKCNVT ICDKDBPR ICDMSSG	Caller
ICDLSSCN	ICDLSSCN		Scans the LIST subcommand; verifies syntax, issues error messages, prompts for new data, and generates code for the subcommand	ICDCHAIN ICDDECHN ICDISCAN ICDMSSG ICDNSCAN ICDPROMT	Caller
ICDMODE	ICDMSSG	ICDATTN ICDSCAN ICDSSCAN	Prints the TEST VSB message		Caller
ICDMSSG	ICDMSSG	ICDADRES ICDATTN ICDCDSCN ICDFLOW ICDGOGO ICDIDCHK ICDISCAN ICDLBKO ICDLISTO ICDLFQO ICDLSSCN ICDOBEY ICDOFFWO ICDONITR ICDON03 ICDON04 ICDON05 ICDPGMCK ICDSCAN ICDSETO ICDSSCAN ICDSSCN	Prints an informational message at the terminal		Caller

Table 4. VS BASIC Debug Module Entry Points (Part 4 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
		ICDSTSCN ICDTBACK ICDTSCN ICDWHENO ICDWHRO ICDWNSCN ICDWNTST ICDVSCN ICDZERO			
ICDMSSGS	ICDMSSGS		Contains pointers to all of the debug messages		
ICDMSTND	ICDMSSGS		End of the no storage message		
ICDMSTXT	ICDMSSGS		Actual text of the debug messages		
ICDMCMDS	ICDCMTBL	ICDDSCAN	Table of subcommand names		
ICDNOSTO	ICDMSSGS		Start of the text of the no storage message		
ICDNSCAN	ICDNSCN	ICDCDSCN	Scans a subcommand buffer until it reaches a delimiter		Caller
ICDOBEY	ICDOBEY	ICDIFOB ICDONITR ICDWNTST	Executes a text element or a chain of text elements for subcommands; this routine also contains the following subroutines: ICDENDO, ICDHALTO, ICDNEXTO, ICDQUALO, and ICDTRCO	ICDATTO ICDDECHN ICDGOGO ICDHELPO ICDLBKO ICDLFQO ICDLISTO ICDMSSG ICDOFFO ICDOFFWO ICDRUNO ICDSETO ICDSTCNV ICDWHENO ICDWHRO	Caller or ICDKRUNX
ICDOCMDS	ICDOBEY		Table of addresses of obey routines		
ICDOFFO	ICDOFFO	ICDOBEY ICDRUNO	Executes the OFF subcommand; removes breakpoint information from the indicated statement	ICDDECHN	Caller
ICDOFFWO	ICDOFFWO	ICDOBEY	Executes the OFFWN subcommand; turns off the monitoring of WHEN conditions	ICDCHAIN ICDDECHN ICDMSSG	Caller
ICDONITR	ICDONITR	ICDDBG	Updates the communications region for the current statement being processed; processes WHEN subcommands, attention interrupts, NEXT subcommands, AT breakpoints; issues trace messages, and updates the statement frequency table	ICDATTN ICDMSSG ICDOBEY ICDSCAN ICDSTCNV ICDWNTST	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 5 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDON02	ICDONITR	ICDDBG2	Monitors transfer of control between program units; called at the entry and exit of user defined functions	ICDCHAIN ICDTSRCH	Caller
ICDON03	ICDONITR	ICDDBG3	Updates the communications region pointers and issues trace messages; called at the entry and exit of user defined functions	ICDDECHN ICDMSSG ICDSTCNV	Caller
ICDON04	ICDONITR	ICDDBG4	Processes the normal termination of a user's program running under debug	ICDMSSG	Caller
ICDON05	ICDONITR	ICDDBG5	Processes an abnormal termination of a user's program running under debug	ICDMSSG	Caller
ICDPGMCK	ICDPGMCK	IKJPARS	Verifies the name of a program unit	ICDMSSG ICDTSRCH	Caller
ICDPMACS	ICDPMACS		Data area containing PCLs that are passed to the TSO routine IKJPARS	IDCHK PGMCHK	Caller
ICDPROMT	ICDMSSG	ICDCDSCN ICDISCAN ICDLSSCN ICDSSCAN ICDSTSCN ICDVSCN ICDWNSCN	Prints the REENTER message		Caller
ICDPRSCN	ICDPRSCN		Scans the following subcommands to determine their operands: NEXT, END, QUALIFY, TRACE, HALT, GO TO, OFF, RUN, OFFWN, HELP, LISTBRKS, and LISTFREQ	IKJPARS ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICDPSCL	ICDPSCL	ICDDSCAN	Processes the AT subcommand	ICDCHAIN ICDDECHN ICDSSCAN IKJPARS TESTYP	Caller
ICDREDIM	ICDREDIM	ICDKRDM1 ICDKRDM2	Redimensions arrays for debug		Caller
ICDRUNO	ICDRUNO	ICDOBEY	Executes the RUN subcommand; removes all AT breakpoints and WHEN conditions, and turns off all tracing	ICDDECHN ICDGOGO ICDOFFO	Caller
ICDSCAN	ICDSCAN	ICDONITR ICDWNTST	Obtains a subcommand from the terminal or attention buffer; verifies the command name and calls the appropriate routines to scan the text	ICDDSCAN ICDMODE ICDMSSG ICDSTCNV	Caller
ICDSCMDS	ICDCMTBL		Table of subcommand names		

Table 4. VS BASIC Debug Module Entry Points (Part 6 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDSETO	ICDSETO	ICDOBEY	Executes the SET subcommand to set a character or numeric variable, array element, or an entire array to a constant, variable, or array element	ICDADRES ICDMSSG	Caller
ICDSSCAN	ICDSSCAN	ICDPSCL	Scans a subcommand list	ICDDSCAN ICDMODE ICDMSSG ICDSTCNV	Caller
ICDSSCN	ICDSSCN	ICDTSCN ICDWNSCN	Scans subscripts	ICDNSCAN ICDPROMT	Caller
ICDSTBL	ICDISCAN		Table for determining if a character is alphabetic or numeric		
ICDSTCNV	ICDSTCNV	ICDFLOW ICDLBKO ICDLFQO ICDOBEY ICDONITR ICDON03 ICDSCAN ICDSSCAN ICDTBACK ICDWNTST ICDWHRO	Formats statement identifiers and program units to be used for message processing		Caller
ICDSTSCN	ICDSTSCN	ICDDSCAN	Scans a subcommand line containing a SET subcommand; verifies the syntax; issues error messages; prompts for data, and produces internal text	ICDDECHN ICDISCAN ICDMSSG ICDNSCAN ICDPROMT	Caller
ICDTBACK	ICDTBACK	ICDWHRO	Prints the traceback as a result of the FUNC or ALL option in the WHERE statement	ICDMSSG ICDSTCNV	Caller
ICDTSCN	ICDTSCN	ICDISCAN	Scans VS BASIC terms	ICDCHAIN ICDCCHN ICDKETOF ICDKETF2 ICDMSSG ICDSSCN ICDNSCAN ICDVSCN	Caller
ICDTSRCH	ICDTSRCH	ICDIDCHK ICDON02 ICDPGMCK ICDVSCN	Searches a program unit directory for a DEF name; searches the statement table for a line number, of searches the symbol table for a symbol		Caller

Table 4. VS BASIC Debug Module Entry Points (Part 7 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDTSTYP	ICDTSTYP	ICDPSCL ICD04SCN ICD05SCN ICD08SCN ICD0ASCN ICDDBSCN ICD0CSCN ICD11SCN ICD12SCN ICD13SCN ICD15SCN ICS16SCN ICD26SCN ICD27SCN	Processes a statement id, a statement id list, or a range of statement ids	ICDCHAIN ICDDECHN	Caller
ICDWHENO	ICDWHENO	ICDOBEY	Executes the WHEN statement	ICDADRES ICDCCHN ICDCHAIN ICDDECHN ICDMSSG	Caller
ICDWHRO	ICDWHRO	ICDATTN	Executes the WHERE subcommand	ICDFLOW ICDMSSG ICDSTCNV ICDTBACK	Caller
ICDWNSCN	ICDWNSCN	ICDCHAIN	Scans the IF and WHEN subcommands	ICDCHAIN ICDCDSCN ICDDECHN ICDISCAN ICDMSSG ICDNSCAN ICDPROMT ICDSSCAN	Caller
ICDWNTST	ICDWNTST	ICDONITR	Determines if any WHEN conditions are satisfied	ICDADRES ICDEVALU ICDMSSG ICDOBEY ICDSCAN ICDSTCNV	Caller
ICDVSCN	ICDVSCN	ICDSSCN ICDTSCN	Scans variable names	ICDMSSG ICDPROMT ICDTSRCH	Caller
ICDZERO	ICDZERO	ICD04SCN ICD05SCN ICD08SCN ICD0ASCN ICD0BSCN ICD0CSCN ICD11SCN ICD12SCN ICD13SCN ICD15SCN ICD16SCN ICD26SCN ICD27SCN	Checks for operands on subcommands that do not require them; prints a message that they are ignored	ICDCHAIN ICDMSSG	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 8 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICD04SCN	ICDPRSCN		Scans the QUALIFY subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD05SCN	ICDPRSCN		Scans the NEXT subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD06SCN			Alias for ICDPSCL		
ICD07SCN			Alias for ICDWNSCN		
ICD08SCN	ICDPRSCN		Scans the OFF subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD0ASCN	ICDPRSCN		Scans the HALT subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD0BSCN	ICDPRSCN		Scans the GO TO subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD0CSCN	ICDPRSCN		Scans the RUN subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD0ESCN			Alias for ICDWNSCN		
ICD0FSCN			Alias for ICDLSSCN		
ICD11SCN	ICDPRSCN		Scans the LISTBRKS subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD12SCN	ICDPRSCN		Scans the LISTFREQ subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD13SCN	ICDPRSCN	ICDCHAIN	Scans the WHERE subcommand	Caller ICDDECHN TESTYP ZEROTXT	
ICD14SCN			Alias for ICDSTSCN		
ICD15SCN	ICDPRSCN		Scans the HELP subcommand	ICDCHAIN ICDDECHN TEXTYP ZEROTXT	Caller

Table 4. VS BASIC Debug Module Entry Points (Part 9 of 9)

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICD16SCN	ICDPRSCN		Scans the END subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD26SCN	ICDPRSCN		Scans the OFFWN subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICD27SCN	ICDPRSCN		Scans the TRACE subcommand	ICDCHAIN ICDDECHN TESTYP ZEROTXT	Caller
ICDCHK			Alias for ICDIDCHK		
PGMCHK			Alias for ICDPGMCK		
TESTYP			Alias for ICDTSTYP		
ZEROTXT			Alias for ICDZERO		

Table 5. VS BASIC Conversion Utility Module Entry Points

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDLUTIL	ICDLUTIL		This module converts BASIC data files in a CALL-OS format to an OS compatible format. (CMS only)	ICDKCNVT	Caller

Table 6. VS BASIC Renumbering Facility Module Entry Points

Entry Points	Module Name	Called By	Function	Calls	Exits To
ICDQRNME	ICDQRNME	IKJEBERE	Provides an interface between the TSO EDIT RENUM routine (IKJEBERRE) and the VS BASIC RENUM routine ICDQRNMS). This routine renumbers specified lines. (TSO only)	ICDQRNMS	IKJEBERE
ICDQRNMS	ICDQRNMS	ICDQRNME or IKJEBMWU via IKJEBMVB	Scans a VS BASIC statement and changes statement number references to correspond with new line numbers. (TSO and CMS)		ICDQRNME

Table 7. VS BASIC Component Directory (Part 1 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
ICDADRES	Module		Debug		ICDADRES
ICDATTN	Module		Debug		ICDATTN
ICDATTO	Module		Debug	9	ICDATTO
ICDBLDTB	Module		Debug	5,9	ICDBLDTB
ICDCDSCN	Module		Debug		ICDCDSAN
ICDCHAIN	Module		Debug	9	ICDCHAIN
ICDCMTBL	Module		Debug		ICDCMTBL
ICDDBG	Module		Debug	9	ICDDBG
ICDDECHN	Entry Point	ICDCHAIN	Debug		ICDCHAIN
ICDDSCAN	Module		Debug	9	ICDDSCAN
ICDEVALU	Module		Debug		ICDEVALU
ICDFLOW	Module		Debug		ICDFLOW
ICDFOSUB	Module		Debug		ICDFOSUB
ICDGOGO	Module		Debug	9	ICDGOGO
ICDHELPO	Module		Debug	9	ICDHELPO
ICDIDCHK	Module		Debug		ICDIDCHK
ICDIFOB	Module		Debug	9	ICDIFOB
ICDISCAN	Module		Debug		ICDISCAN
ICDJAADJ	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJALOC	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJCDEF	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR
ICDJCEND	Entry Point	ICDJNUCL	Compiler	3	ICDJNUCL
ICDJCHN	Entry Point	ICDJVERB	Compiler		ICDJVERB
ICDJCLOS	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJCMPA	Module		Compiler	3	ICDJCMPA
ICDJCNVT	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJCONF	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJCTL	Entry Point	ICDJNUCL	Compiler	3,4	ICDJNUCL
ICDJDATA	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR
ICDJDDAT	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJDEFR	Module		Compiler	3	ICDJDEFR
ICDJDEF1	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJDEF2	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJDEXT	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJDFRM	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJDIM	Entry Point	ICDJVERB	Compiler		ICDJVERB
ICDJDIMG	Entry Point	ICDJNUCL	Compiler	3,4	ICDJNUCL
ICDJEND	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR
ICDJERR	Module		Compiler	3,4	ICDJERR
ICDJERRN	Entry Point	ICDJERR	Compiler		ICDJERR
ICDJERRP	Entry Point	ICDJERR	Compiler		ICDJERR
ICDJERRS	Entry Point	ICDJERR	Compiler	4	ICDJERR
ICDJERRT	Entry Point	ICDJERR	Compiler		ICDJERR
ICDJEXIT	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR

Table 7. VS BASIC Component Directory (Part 2 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
ICDJFBN1	Entry Point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFBN2	Entry Point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFCAT	Entry Point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFEXT	Entry point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFGEN	Entry Point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFMLA	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJFNE1	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJFNE2	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJFOR	Entry Point	ICDJNUC2	Compiler		ICDJNUC2
ICDJFORM	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR
ICDJFRM2	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJFRM3	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJFRM5	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJFUNY	Entry Point	ICDJFUTS	Compiler		ICDJFUTS
ICDJFUTS	Module		Compiler		ICDJFUTS
ICDJGET	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJGOSB	Entry Point	ICDJNUC4	Compiler		ICDJNUC4
ICDJGOTO	Entry Point	ICDJNUC4	Compiler		ICDJNUC4
ICDJIF	Entry Point	ICDJNUC5	Compiler		ICDJNUC5
ICDJIF1	Entry Point	ICDJNUC5	Compiler		ICDJNUC5
ICDJIF2	Entry Point	ICDJNUC5	Compiler		ICDJNUC5
ICDJIMAG	Entry Point	ICDJDEFR	Compiler	3	ICDJDEFR
ICDJINFO	Module		Compiler		ICDJINFO
ICDJINPT	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJIOVB	Module		Compiler	4	ICDJIOVB
ICDJLET	Entry Point	ICDJNUC2	Compiler		ICDJNUC2
ICDJLINE	Entry Point	ICDJNUCL	Compiler	4	ICDJNUCL
ICDJMATD	Entry Point	ICDJMATV	Compiler		ICDJMATV
ICDJMATV	Module		Compiler	4	ICDJMATV
ICDJNEXT	Entry Point	ICDJNUC3	Compiler		ICDJNUC3
ICDJNUCL	Module		Compiler	3,4	ICDJNUCL
ICDJNUC1	Module		Compiler		ICDJNUC1
ICDJNUC2	Module		Compiler	4	ICDJNUC2
ICDJNUC3	Module		Compiler	4	ICDJNUC3
ICDJNUC4	Module		Compiler	4	ICDJNUC4
ICDJNUC5	Module		Compiler	4	ICDJNUC5
ICDJON	Entry Point	ICDJVERB	Compiler		ICDJVERB
ICDJOPEN	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJPAUS	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJPRNT	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJPUT	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJRDIM	Entry Point	ICDJMATV	Compiler		ICDJMATV
ICDJREAD	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJRETN	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJRETV	Entry Point	ICDJUSFN	Compiler		ICDJUSFN
ICDJRSET	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJRSTO	Entry Point	ICDJIOVB	Compiler		ICDJIOVB
ICDJRUNA	Module		Compiler	3	ICDJRUNA
ICDJSCN1	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJSCN2	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJSTOP	Entry Point	ICDJVERB	Compiler		ICDJVERB
ICDJTFN	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJTFOR	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJTRD	Entry Point	ICDJNUCL	Compiler		ICDJNUCL
ICDJUSE	Entry Point	ICDJVERB	Compiler		ICDJVERB
ICDJUSFN	Module		Compiler	4	ICDJUSFN

Table 7. VS BASIC Component Directory (Part 3 of 8)

Name	Type	Module	Component	MO Diagram	Microtiche
ICDJVAL1	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJVAL2	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJVAL3	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJVAL4	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJVAL5	Entry Point	ICDJNUC1	Compiler		ICDJNUC1
ICDJVABL	Entry Point	ICDJVREC	Compiler		ICDJVEXC
ICDJVERB	Module		Compiler	4	ICDJVERB
ICDJVREC	Module		Compiler	4	ICDJVREC
ICDJVRD	Entry Point	ICDJVREC	Compiler		ICDJVREC
ICDJVRRD	Entry Point	ICDJVREC	Compiler		ICDJVREC
ICDJVRWR	Entry Point	ICDJVREC	Compiler		ICDJVREC
ICDJVWRT	Entry Point	ICDJVREC	Compiler		ICDJVREC
ICDKACS	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKASN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKATN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKBFTB	Module		Library		ICDKBFTB
ICDKCHN	Entry Point	ICDKERR	Library		ICDKERR
ICDKCLK	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKCLOS	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKCNVT	Module		Library	6	ICDKCNVT
ICDKCOS	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKCOT	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKCPU	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKCSC	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKDABS	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDACS	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDASN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDAT	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKDATN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDBPR	Entry Point	ICDKPRNT	Library		ICDKPRNT
ICDKDCOS	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDCOT	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDCSC	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDET	Entry Point	ICDKMINV	Library		ICDKMINV
ICDKDEXP	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDHCS	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDHSN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDHTN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDLGT	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDLOG	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDLTW	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDMAX	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDMIN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDOT	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKDPWR	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDSEC	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDSIN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDSQR	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKDSUB	Module		Library	5	ICDKDSUB
ICDKDTAN	Entry Point	ICDKDSUB	Library		ICDKDSUB
ICDKETAB	Entry Point	ICDKETOF	Library		ICDKETOF
ICDKERR	Module		Library	5	ICDKERR
ICDKERRR	Entry Point	ICDKERR	Library	5	ICDKERR
ICDKERRS	Entry Point	ICDKERR	Library	5,6	ICDKERR
ICDKERRT	Entry Point	ICDKERR	Library	5	ICDKERR
ICDKETF2	Entry Point	ICDKETOF	Library	7	ICDKETOF
ICDKETOF	Module		Library	6,7	ICDKETOF
ICDKEXP	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKFSCN	Entry Point	ICDKIOVB	Library	6,8	ICDKIOVB
ICDKGET	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKGSUB	Module		Library	5	ICDKGSUB
ICDKHCS	Entry Point	ICDKSSUB	Library		ICDKSSUB

Table 7. VS BASIC Component Directory (Part 4 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
ICDKHSN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKHTN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKIDX	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKINPT	Module		Library	5,7	ICDKINPT
ICDKINTP	Module		Library	5	ICDKINTP
ICDKIOVB	Module		Library	5,6	ICDKIOVB
ICDKJDY	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKKLN	Module		Library		ICDKKLN
ICDKKPS	Module		Library		ICDKKPS
ICDKLEN	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKLGT	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKLOG	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKLTW	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKMADD	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMASN	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMASR	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMAT	Module		Library		ICDKMAT
ICDKMAX	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKMDSR	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMIDN	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMIN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKMINV	Module		Library		ICDKMINV
ICDKMMUL	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMSCA	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMSUB	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKMTRN	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKNCPD	Module		Library		ICDKNCPD
ICDKNUM	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKON	Entry Point	ICDKERR	Library	5	ICDKERR
ICDKOPEN	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKOPN1	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKORGE	Module		Library	2,5,9	ICDKORGE
ICDKPLIN	Module		Library	7	ICDKPLIN
ICDKPRD	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKPRNT	Module		Library	7	ICDKPRNT
ICDKPUT	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKPWR	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKRDM1	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKRDM2	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKREAD	Module		Library		ICDKREAD
ICDKRLN	Module		Library		ICDKRLN
ICDKRND	Entry Point	ICDKSSUB	Library		ICDKRND
ICDKRSET	Entry Point	ICDKIOVB	Library	6	ICDKIOVB
ICDKRUNX	Entry Point	ICDKERR	Library	3,5	ICDKERR
ICDKRUNY	Entry Point	ICDKERR	Library	5	ICDKERR
ICDKSEC	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKSIN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKSQR	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKSSUB	Module		Library	5	ICDKSSUB
ICDKSTR	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKSUM	Entry Point	ICDKMAT	Library		ICDKMAT
ICDKTAN	Entry Point	ICDKSSUB	Library		ICDKSSUB
ICDKTIM	Entry Point	ICDKGSUB	Library		ICDKGSUB
ICDKTIO	Entry Point	ICDKPRNT	Library	7	ICDKPRNT
ICDKTOUT	Module		Library	7	ICDKTOUT
ICDKVCLS	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVDEL	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVEND	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVIOR	Module		Library	5,8	ICDKVIOR
ICDKVOPN	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVRD	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVRST	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVRRD	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVRWR	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVTIO	Entry Point	ICDKVIOR	Library	8	ICDKVIOR
ICDKVWRT	Entry Point	ICDKVIOR	Library	8	ICDKVIOR

Table 7. VS BASIC Component Directory (Part 5 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
ICDLBKO	Module		Debug	9	ICDKLBKO
ICDLFQO	Module		Debug	9	ICDLFQO
ICDLISTO	Module		Debug	9	ICDLISTO
ICDLSSCN	Module		Debug	9	ICDLSSCN
ICDLUTIL	Module		Utility	11	ICDLUTIL
ICDMODE	Entry Point	ICDMSSG	Debug	9	ICDMSSG
ICDMSSG	Module		Debug	9	ICDMSSG
ICDMSSGS	Module		Debug		ICDMSSGS
ICDNAMLO	Module		Debug		ICDNAMLO
ICDNSCAN	Module		Debug		ICDNSCAN
ICDOBEY	Module		Debug	9	ICDOBEY
ICDOFFO	Module		Debug	9	ICDOFFO
ICDOFFWO	Module		Debug	9	ICDOFFWO
ICDONITR	Module		Debug	9	ICDONITR
ICDON02	Entry Point	ICDONITR	Debug		ICDONITR
ICDON03	Entry Point	ICDONITR	Debug		ICDONITR
ICDSMCK	Module		Debug		
ICDPCLS	Module		Executor	8	ICDPCLS
ICDPEXEC	Module		VSPC Executor	1-8	ICDPEXEC
ICDPOPN	Module		Executor	8	ICDPOPN
ICDPPENT	Module		Executor	8	ICDPPENT
ICDPRSCN	Module		Debug	9	ICDPRSCN
ICDPSCL	Module		Debug	9	ICDPSCL
ICDPURGO	Module		Debug		ICDPURGO
ICDPVENT	Module		Executor	8	ICDPVENT
ICDQEXEC	Module		TSO	1-8	ICDQEXEC
			Executor		
ICDQRNME	Module		RENUM	10	ICDQRNME
ICDQRNMS	Module		RENUM	10	ICDQRNMS
ICDQVCLS	Module		VSAM		ICDQVCLS
ICDQVDEL	Module		VSAM		ICDQVDEL
ICDQVENT	Module		VSAM		ICDQVENT
ICDQVERR	Module		VSAM		ICDQVERR
ICDQVGET	Module		VSAM		ICDQVGET
ICDQVOPN	Module		VSAM		ICDQVOPN
ICDQVPNT	Module		VSAM		ICDQVPNT
ICDQVPUT	Module		VSAM		ICDQVPUT
ICDQWCLS	Entry Point	ICDQWCLS	Executor		ICDQWCLS
ICDQWDEL	Entry Point	ICDQWDEL	Executor		ICDQWDEL
ICDQWENT	Entry Point	ICDQWENT	Executor		ICDQWENT
ICDQWERR	Entry Point	ICDQWERR	Executor		ICDQWERR
ICDQWGET	Entry Point	ICDQWGET	Executor		ICDQWGET
ICDQWOPN	Entry Point	ICDQWOPN	Executor		ICDQWOPN
ICDQWPNT	Entry Point	ICDQWPNT	Executor		ICDQWPNT
ICDQWPUT	Entry Point	ICDQWPUT	Executor		ICDQWPUT
ICDQZCLS	Module		DOS VSAM		ICDQZCLS
ICDQZDEL	Module		DOS VSAM		ICDQZDEL
ICDQZENT	Module		DOS VSAM		ICDQZENT
ICDQZERR	Module		DOS VSAM		ICDQZERR
ICDQZGET	Module		DOS VSAM		ICDQZGET
ICDQZOPN	Module		DOS VSAM		ICDQZOPN
ICDQZPNT	Module		DOS VSAM		ICDQZPNT
ICDQZPUT	Module		DOS VSAM		ICDQZPUT
ICDRUNO	Module		Debug	9	ICDRUNO
ICDSCAN	Module		Debug	9	ICDSCAN
ICDSETO	Module		Debug	9	ICDSETO
ICDSSCAN	Module		Debug		ICDSSCAN
ICDSTCNV	Module		Debug		ICDSTCNV
ICDSTSCN	Module		Debug	9	ICDSTSCN
ICDTBACK	Module		Debug		ICDTBACK
ICDTMSG	Entry Point	ICDMSSG	Debug		ICDMSSG
ICDTSCN	Module		Debug		ICDTSCN
ICDTSRCH	Module		Debug		ICDYSRCH

Table 7. VS BASIC Component Directory (Part 6 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
ICDWEXEC	Module		CMS Executor	1-8	ICDWEXEC
ICDTSTYP	Module		Debug		ICDTSTYP
ICDVSCN	Module		Debug		ICDVSCN
ICDWHENO	Module		Debug	9	ICDWHENO
ICDWHRO	Module		Debug	9	ICDWHRO
ICDWNSCN	Module		Debug		ICDWNSCN
ICDWNTST	Module		Debug	9	ICDWNTST
ICDYEXEC	Module		OS/VS Executor	1-8	ICDYEXEC
ICDZEXEC	Module		DOS/VS Executor	1-8	ICDZEXEC
ICD0ASCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD0BSCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD0CSCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD03SCN	Entry Point	ICDPRSCN	Debug		ICDPRSCN
ICD04SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD05SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD07SCN	Entry Point	ICDPRSCN	Debug		ICDPRSCN
ICD08SCN	Entry Point	ICDPRSCN	Debug	9	
ICD09SCN	Entry Point	ICDPRSCN	Debug		ICDPRSCN
ICD11SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD12SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD13SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD15SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD16SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD24SCN	Entry Point	ICDPRSCN	Debug		ICDPRSCN
ICD26SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
ICD27SCN	Entry Point	ICDPRSCN	Debug	9	ICDPRSCN
SPIEEXIT	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDZEXEC			ICDZEXEC
STAEEXIT	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDZEXEC			ICDZEXEC
STAXEXIT	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
SVC0	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC1	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC2	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC3	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC4	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			

Table 7. VS BASIC Component Directory (Part 7 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
		ICDZEXEC			ICDZEXEC
SVC5	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC6	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC7	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC8	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC10	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC11	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC12	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC13	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC14	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC16	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC18	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC21	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC22	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC

Table 7. VS BASIC Component Directory (Part 8 of 8)

Name	Type	Module	Component	MO Diagram	Microfiche
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC23	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDWEXEC			ICDWEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC24	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC25	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC
SVC26	Entry Point	ICDQEXEC	Executor		ICDQEXEC
		ICDYEXEC			ICDYEXEC
		ICDPEXEC			
		ICDZEXEC			ICDZEXEC

DATA AREAS

This section describes the data areas used by the VS BASIC Processor. Table 8 serves as an index to the data areas. It lists each label and the data area in which it is located

Table 8. Data Area Directory (Part 1 of 14)

NAME	DATA AREA
$0	NUC
$1	NUC
$2	NUC
$3	NUC
$4	NUC
$5	NUC
$6	NUC
$7	NUC
$8	NUC
$9	NUC
$10	NUC
$11	NUC
$12	NUC
$16	NUC
$17	NUC
$18	NUC
$22	NUC
$23	NUC
$24	NUC
$29	NUC
$30	NUC
$36	NUC
$38	NUC
$39	NUC
$60	NUC
$90	NUC
$100	NUC
$200	NUC
$255	NUC
$256	NUC
$1000	NUC
$2048	NUC
$4095	NUC
$4096	NUC
$32767	NUC
$64000	NUC
$AD	PRG(C)
$ADR	PRG(C)
$AE	PRG(C)
$AER	PRG(C)
$AU	PRG(C)
$AUR	PRG(C)
$AW	PRG(C)
$AWR	PRG(C)
$CD	PRG(C)
$CDR	PRG(C)
$CE	PRG(C)
$CER	PRG(C)
$DD	PRG(C)
$DDR	PRG(C)
$DE	PRG(C)
$DER	PRG(C)
$HDR	PRG(C)
$HER	PRG(C)
$LCDR	PRG(C)
$LCER	PRG(C)
$LD	PRG(C)
$LDR	PRG(C)
$LE	PRG(C)
$LER	PRG(C)
$LNDR	PRG(C)
$LNER	PRG(C)

Table 8. Data Area Directory (Part 2 of 14)

NAME	DATA AREA
$LPDR	PRG(C)
$LPER	PRG(C)
$LTDR	PRG(C)
$LTER	PRG(C)
$MD	PRG(C)
$MDR	PRG(C)
$ME	PRG(C)
$MER	PRG(C)
$SD	PRG(C)
$SDR	PRG(C)
$SE	PRG(C)
$SER	PRG(C)
$STD	PRG(C)
$STE	PRG(C)
$SU	PRG(C)
$SUR	PRG(C)
$SW	PRG(C)
$SWR	PRG(C)
AADJAR	PRG(C)
AARJPR	PRG
ABSWORK	PRG(C)
ARGADDR	ARGENTRY
ARGCLTH	ARGENTRY
ARGCODE	ARGENTRY
ARGERR	OBJAREA
ARGOPNCD	ARGENTRY
ARGPRCD	ARGENTRY
ARGRSCD	ARGENTRY
ARINTRP	PRG
ARRBYT	PRG(C)
ARRSVC	PRG(C)
ARRSVP	PRG
ARYDSCD1	ARGENTRY
ARYDSCD2	ARGENTRY
ARYDSCL	ARGENTRY
ARYDSCMY	ARGENTRY
ARYDSCND	ARGENTRY
ARYDSCOS	ARGENTRY
ASLADD	PRG(C)
ASLMASK	PRG(C)
ASLTH	PRG
ASMAX	PRG
ASPTR	PRG
ASVCCTL	PRG
ATTNFLAG	PRG
AVEXT	PRG(C)
AVFOR	PRG(C)
AVIM	PRG(C)
AVLIN	PRG
AVVAR	PRG(C)
BOT28	NUC
BOT29	NUC
B13	NUC
B13A15	NUC
B14	NUC
B14T15	NUC
B15	NUC
B16T31	NUC
B20T31	NUC
B29T31	NUC
BASPROC	PRG
BASUSER	PRG
BIFTAB	ICDBIFTB

Table 8. Data Area Directory
(Part 3 of 14)

NAME	DATA AREA
BLKPAD	PRG(R)
BLOCKMAX	FILTAB
BRP	PRGA
BSDAT	PRG
BSEXTPTR	PRG
BSKEY	PRG
BSLFORM	PRG
BSLINCHN	PRG
BSLINTAB	PRG
BSLINTB	PRG
BSLNPTRS	PRG
BSOBJ	PRG
BSOVFLW	PRG
BSPARMSV	PRG
BSPRG	PRG
BSPRGA	PRG
BSSRC	PRG
BSUFUN	PRG
BSVARCN1	PRG
BSVARCN2	PRG
BSVARPTR	PRG
BUCKET1	PRG(R)
BUFDIS	FILTAB
BUFF	VARCON
BUFFAHED	PRG
BUFFDLIM	PRG(R)
BUFFLNTH	PRG(R)
BUFFRRA0	PRG(R)
BUFFRRA4	PRG(R)
BUFFSTRT	PRG(R)
BUFFWRK	PRG(R)
BUFLTH	PRG
BUFPTR	PRG
CALLER	PRG(R)
CALLNGTH	PRG(R)
CHAR1	PRGA
CHAR2	PRGA
CNOERMSG	PRG(C)
CNOLINE	PRG(C)
CNVTOUT	PRG(R)
CNVTRET	PRG(R)
COMATNEB	COMREGN
COMATNLN	COMREGN
COMCASTB	COMREGN
COMCNVTA	COMREGN
COMCPPL	COMREGN
COMCURPU	COMREGN
COMCURST	COMREGN
COMDIRCN	COMREGN
COMFMTA	COMREGN
COMENDEB	COMREGN
COMFLAGS	COMREGN
COMFLOW	COMREGN
COMIFMTA	COMREGN
COMJUMP	COMREGN
COMKPARP	COMREGN
COMKSCNP	COMREGN
COMLASST	COMREGN
COMLSTDR	COMREGN
COMLSTPU	COMREGN
COMNASTB	COMREGN
COMNVSTB	COMREGN

Table 8. Data Area Directory
(Part 4 of 14)

NAME	DATA AREA
COMOASTB	COMREGN
COMPABS	PRG
COMPB	COMREGN
COMPPTGT	COMREGN
COMPUTLP	COMREGN
COMQUAL	COMREGN
COMSTAB	COMREGN
COMSUBST	COMREGN
COMTRACE	COMREGN
COMWHNCN	COMREGN
COMZFLGS	COMREGN
CONCAT1	OBJAREA
CONSVA	PRGA
CONSVC	PRGA
CONVENT	CNDTBL
CONVERSW	ESPACE
CPPLADDR	PRG
CSTART	PRG
CTAB	NUC
CTLSV1	PRG
CTLSV2	PRG
CURBSREG	PRG(C)
CURCTL	PRG(C)
CURDAT	PRG
CURDEF	PRG(C)
CURIN	FILTAB
CURKEY	PRG
CURLINE	PRG(C)
CURRCI	VFILTAB
DATAKEYS	PRG(C)
DATAWORD	ESPACE
DATE	PRG
DATUM	PRGA
DBUGFLGS	PRG
DBUGNTRY	PRG
DBUGRSRV	PRG
DEBUGTAB	PRG
DEFNAME	PRG(C)
DEFORCT	PRG(C)
DEFSAV	PRG
DIGIT	PRGA
DIRCHAIN	DIR
DIREXSYM	DIR
DIRNAME	DIR
DKBFSIZE	NUC
DTEMP	ESPACE
DUPKYENT	CNDTBL
E	VARCON
EACBA	ESPACE
EACTSEQ	ESPACE
EAREA	ESPACE
EAREALN	ESPACE
ECKEYA	ESPACE
ECLSOPEN	ESPACE
ECURACT	ESPACE
EFILENM	ESPACE
EFILENO	ESPACE
EFMT	PRG(R)
EFORG	ESPACE
EFREE	ESPACE
EKEYINCR	ESPACE
ENAMELN	ESPACE
ENDDAT	PRG

Table 8. Data Area Directory (Part 5 of 14)

NAME	DATA AREA
ENDLIN	PRG(C)
ENDVAR	PRG(C)
EOFENT	CNDTBL
EOPEN	ESPACE
EOPNMODE	ESPACE
EOPNSTAT	ESPACE
EPARAMS	ESPACE
EPLINCR	ESPACE
ERABS	PRG(R)
EREQST	ESPACE
ERETNCOD	ESPACE
ERKP	ESPACE
ERMFLAG	PRG
ERRC	VARCON
ERRCODE	ESPACE
ERRF	VARCON
ERRL	VARCON
ERRN	VARCON
ERROCCUR	PRG(C)
ERSVERRL	PRG(R)
ERSVRET	PRG(R)
ERSVRET2	PRG(R)
ETYPE	ESPACE
EVEXCEPT	ESPACE
EXECSWIT	FILTAB
EXITDISP	FILTAB
EXITDSP	ESPACE
EXPON	PRGA
EXSW	PRG(R)
EXTDISP	EXTPTRS
EXTEND	PRG
EXTNUMS	PRGA
FCNMRK	PRG
FENDERR	OBJAREA
FFMT	PRG(R)
FILEMBR	PRG
FILENAME	FILTAB
FILENUM	FILTAB
FILEPTR	ORG
FILE2K	PRG
FILLCHAR	ESPACE
FIX255	VARCON
FLTFXM1	VARCON
FLTMIN1	VARCON
FLTPI180	VARCON
FLTPLUS1	VARCON
FLT0	VARCON
FLT1	VARCON
FLT2	VARCON
FLT3	VARCON
FLT4	VARCON
FLT5	VARCON
FLT5S9	VARCON
FLT6	VARCON
FLT7	VARCON
FLT8	VARCON
FLT9	VARCON
FLT9S5	VARCON
FLT32	VARCON
FLT47	VARCON
FLT180PI	VARCON
FLT32767	VARCON
FMLASVA	PRGA

Table 8. Data Area Directory (Part 6 of 14)

NAME	DATA AREA
FMLASVC	PRGA
FMLASVP	PRG
FMTFLG	PRG(R)
FORMINCR	ESPACE
FORSTACK	PRGA
FORTMP	PRG
FPHCFLAG	PRG
FPTEMP	PRG(R)
FRSTBYTE	FILTAB
FSTDAT	PRG(C)
FSTDIM	PRG(C)
FSTEXT	PRG(C)
FSTERM	PRG(C)
FTRK2K	PRG
FTYPE	FILTAB
GAL1	VARCON
GENFLAG	PRG(C)
GOSUBER	OBJAREA
GOTOTMP	PRG(R)
GOTOTMP2	PRG(R)
HASPTRS	PRGA
HDRCDDRG	PRG(C)
HDRCDSTD	PRG(C)
HIREC#	VFILTAB
HIWRITE	ESPACE
H4	PRG(R)
H8	PRG(R)
IBFPTR	PRG
IDSVRC3	PRGA
IFMT	PRG(R)
IFRTMP0	PRG
IFRTMP1	PRG
IFRTMP2	PRG
IFTMP	PRG
IMEND	PRG
IMNUMS	PRGA
INCM	VARCON
INFOBIF	INFOTAB
INFOCD	INFOTAB
INFOFLG	INFOTAB
INFONAME	INFOTAB
INFONXT	INFOTAB
INFORESV	INFOTAB
INPRES	FILTAB
INPUTFNO	PRG(R)
INTCON	VARCON
IOCNT	PRG(R)
IOCODE	FILTAB
IOERRENT	CNDTBL
ITEMCNT	FILTAB
JDYTOD	PRG(R)
KROUTRET	PRG(R)
LADDRESS	ESPACE
LASTREC#	VFILTAB
L#2EB	Z#UTT
L#FLGS	Z#UTT
L#LANG	Z#UTT
L#N2048	Z#UTT
L#NLINE	Z#UTT
L#OLANG	Z#UTT
L#RTINE	Z#UTT
L#SADDR	Z#UTT
L#SOURC	Z#UTT

Table 8. Data Area Directory
(Part 7 of 14)

NAME	DATA AREA
L#WIDTH	Z#UTT
LBKG	VARCON
LCCDEF	NUC
LCCHAIN	NUC
LCCLOSE	NUC
LCCMPA	NUC
LCCTL	NUC
LCDATA	NUC
LCDDAT	NUC
LCDEF2	NUC
LCDEF	NUC
LCDELETE	NUC
LCDFRM	NUC
LCDIM	NUC
LCDIMAG	NUC
LCEND	NUC
LCERRN	NUC
LCERRP	NUC
LCERRS	NUC
LCERRT	NUC
LCEXIT	NUC
LCFBN1	NUC
LCFBN2	NUC
LCFCAT	NUC
LCFEXP	NUC
LCFGEN	NUC
LCFNE1	NUC
LCFNE2	NUC
LCFOR	NUC
LCFORM	NUC
LCFRM2	NUC
LCFRM3	NUC
LCRFM5	NUC
LCFUNY	NUC
LCGET	NUC
LCGOSUB	NUC
LCGOTO	NUC
LCIF1	NUC
LCIF2	NUC
LCIF	NUC
LCIMAG	NUC
LCINFO	NUC
LCINPUT	NUC
LCKACS	ICDBIFTB
LCKASN	ICDBIFTB
LCKATN	ICDBIFTN
LCKATTN	ICDBIFTB
LCKCHN	ICDBIFTB
LCKCHR	ICDBIFTB
LCKCLK	ICDBIFTB
LCKCLOS	ICDBIFTB
LCKCNVT	ICDBIFTB
LCKCO	ICDBIFTB
LCKCOT	ICDBIFTB
LCKCPU	ICDBIFTB
LCKCSC	ICDBIFTB
LCKDACS	ICDBIFTB
LCKDASN	ICDBIFTB
LCKDAT	ICDBIFTB
LCKDCOS	ICDBIFTB
LCKDCOT	ICDBIFTB
LCKDCSC	ICDBIFTB
LCKDEXP	ICDBIFTB

Table 8. Data Area Directory
(Part 8 of 14)

NAME	DATA AREA
LCKDHCS	ICDBIFTB
LCKDHSN	ICDBIFTB
LCKDHTN	ICDBIFTB
LCKDLGT	ICDBIFTB
LCKDLOG	ICDBIFTB
LCKDLTW	ICDBIFTB
LCKDMAX	ICDBIFTB
LCKAMIN	ICDBIFTB
LCKDOT	ICDBIFTB
LCKDPWR	ICDBIFTB
LCKDSEC	ICDBIFTB
LCKDSIN	ICDBIFTB
LCKDSQR	ICDBIFTB
LCKDTAN	ICDBIFTB
LCKERRR	ICDBIFTB
LCKERRS	ICDBIFTB
LCKERRT	ICDBIFTB
LCKETF2	ICDBIFTB
LCKETF3	ICDBIFTB
LCKETF4	ICDBIFTB
LCKETOF	ICDBIFTB
LCKFSCN	ICDBIFTB
LCKGET	ICDBIGTB
LCKHCS	ICDBIFTB
LCKHSN	ICDBIFTB
LCKHTN	ICDBIFTB
LCKIDX	ICDBIFTB
LCKINPT	ICDBIFTB
LCKINTP	ICDBIFTB
LCKJDY	ICDBIFTB
LCKKLN	ICDBIFTB
LCKKPS	ICDBIFTB
LCKLEN	ICDBIFTB
LCKLGT	ICDBIFTB
LCKLOG	ICDBIFTB
LCKLTW	ICDBIFTB
LCKMADD	ICDBIFTB
LCKMASN	ICDBIFTB
LCKMASR	ICDBIFTB
LCKMAX	ICDBIFTB
LCKMDSR	ICDBIFTB
LCKMIDN	ICDBIFTB
LCKMIN	ICDBIFTB
LCKMINV	ICDBIFTB
LCKMMUL	ICDBIFTB
LCKMSCA	ICDBIFTB
LCKMSUB	ICDBIFTB
LCKMTRN	ICDBIFTB
LCKNUM	ICDBIFTB
LCKON	ICDBIFTB
LCKOPEN	ICDBIFTB
LCKORGE	ICDBIFTB
LCKPLIN	ICDBIFTB
LCKPRD	ICDBIFTB
LCKPWR	ICDBIFTB
LCKRDM1	ICDBIFTB
LCKRDM2	ICDBIFTB
LCKREAD	ICDBIFTB
LCKRLN	ICDBIFTB
LCKRND	ICDBIFTB
LCKRSET	ICDBIFTB
LCKRUNX	ICDBIFTB
LCKSEC	ICDBIFTB

Table 8. Data Area Directory (Part 9 of 14)

NAME	DATA AREA
LCKSIN	ICDBIFTB
LCKSQR	ICDBIFTB
LCKSUM	ICDBIFTB
LCKTAN	ICDBIFTB
LCKTIM	ICDBIFTB
LCKTIO	ICDBIFTB
LCKTOUT	ICDBIFTB
LCKVCLS	ICDBIFTB
LCKVDEL	ICDBIFTB
LCKVEXT	ICDBIFTB
LCKVFND	ICDBIFTB
LCKVNXT	ICDBIFTB
LCKVOPN	ICDBIFTB
LCKVRD	ICDBIFTB
LCKVRRD	ICDBIFTB
LCKVRST	ICDBIFTB
LCKVRWR	ICDBIFTB
LCKVTIO	ICDBIFTB
LCKVWRT	ICDBIFTB
LCLET	NUC
LCMAT	NUC
LCMATD	NUC
LCNEXT	NUC
LCNUC1	NUC
LCOPEN	NUC
LCPAUSE	NUC
LCPRINT	NUC
LCPUT	NUC
LCRDIM	NUC
LCREAD	NUC
LCREREAD	NUC
LCRESET	NUC
LCRESTOR	NUC
LCRETURN	NUC
LCRETV	NUC
LCREWRIT	NUC
LCRUNA	NUC
LCSRCH	NUC
LCSTART	ICDBIFTB
LCSTOP	NUC
LCTFN	NUC
LCTRD	NUC
LCTRD	NUC
LCUSE	NUC
LCUSTB	NUC
LCVRD	NUC
LCWRITEF	NUC
LETTMP	PRG
LEV0ABS	PRG(R)
LCWRITEF	NUC
LETTMP	PRG
LEV0ABS	PRG(R)
LEV1ABS	PRG(R)
LEV3ABS	PRG(R)
LEVM1ABS	PRG(R)
LPORTMP	NUC
LIFTMP	NUC
LINCHN1	LINTAB
LINCHN2	LINTAB
LINCHN3	LINCHN
LINFNO	LINTAB
LINOBJ	LINPTRS
LINSAV	PRGA

Table 8. Data Area Directory (Part 10 of 14)

NAME	DATA AREA
LINSRCE	LINTAB
LINSVA	PRGA
LINSVP	PRG
LINUM1	LINPTRS
LINUM2	LINCHN
LINWDTH	PRG
LOCBRANC	PRG
LOCWORD	ESPACE
LRECLMAX	FILTAB
MASK1	PRG(R)
MASK3	PRG(P)
MASKM1	PRG(R)
MASKM2	PRG(R)
MATDABS	PRG(C)
MATDRC1	PRG(C)
MATDRC3	PRG(C)
MATMP1	PRG
MATMP2	PRG
MATRPSV	PRG
MAXADDR	ESPACE
MAXDGTS	PRG(R)
MAXFILES	PRG
MAXJSIZE	PRG
MAXNBLIN	PRG(R)
MIDCLK	PRG(R)
MODE	PRG(C)
MODEFLAG	FILTAB
MSGCNT	PRG
NAMADDR	ICDNAME
NAMDIMS	ICDNAME
NAMEOT	ICDNAME
NAMLEN	ICDNAME
NAMLNGTH	FILTAB
NAMMODE	ICDNAME
NAMNAME	ICDNAME
NAMNODIM	ICDNAME
NAMTYPE	ICDNAME
NBLIN	PRG(R)
NEEDBUF	PRG
NMOVE	ESPACE
NOERMSG	PRG
NOKEYENT	CNDTBL
NOLINE	PRG(C)
NULLSTR	VARCON
OBJPTR	PRG
ONATTN	PRG(R)
ONCELLS	PRG(R)
ONERR	PRG(R)
ONINERR	PRG(R)
ONOFLOW	PRG(R)
ONTARGET	PRG(R)
ONUFLOW	PRG(R)
ONZDIV	PRG(R)
OPDS	PRGA
OPENFLG	FILTAB
OPFLG	PRG
OPRS	PRGA
OPTIONS	PRG
PADCHAR	PRG(R)
PARMCNT	PRG(C)
PARMPTR	PRG
PARMTBL	PRGA
PARMCNT	PRG(C)

Table 8. Data Area Directory
(Part 11 of 14)

NAME	DATA AREA
PARMPTR	PRG
PARMTBL	PRGA
PDMPBGN	PRG
PDMPEND	PRG
PHYSBUF	FILTAB
PHYSEND	FILTAB
PI	VARCON
PLINE	PRG(R)
PRBGN	PRG
PRINTFNO	PRG(R)
PSLTH	PRG
PSMAX	PRG
PSMIN	PRG
PSREG	PRG
PSSAV	PRG
PSW1SV	PRG
PSW2SV	PRG
QTCHAR	PRG(C)
RDABS	PRG(C)
RDECNO1	PRG(R)
RDECNO2	PRG(R)
RDIMRC1	PRG(C)
RDIMRC3	PRG(C)
RDIM1	PRG(R)
RDIM2	PRG(R)
RECCNT	FILTAB
RECLNTH	VFILTAB
RECOGNIZ	PRG
RECRDEND	FILTAB
REDIM	PRG(C)
REFERR1	IBJAREA
REFERR2	IBJAREA
RELREC#	VARCON
RELWORK	PRG
RESETFLG	FILTAB
RETSW	PRG(C)
RETSWSAV	PRGA
RETURN	ESPACE
RETURNER	OBJAREA
RNDSEED	PRG(R)
RNDTMP	PRG
RUNPARM	PRG
RUNPARMD	PRG
RUNPARML	PRG
SAVEBASE	PRG
SAVER0	PRG
SAVER15	PRG
SAVESTOR	PRG
SAVFORMA	ESPACE
SAVREG	PRG
SAVRLOC	ESPACE
SAVSWTCH	PRG
SCNCODE	PRGA
SCNIRP3	PRG
SCNTMP	PRGA
SCN1RA2	PRGA
SCN1RA3	PRGA
SCN1RC2	PRGA
SCN1RP1	PRG
SCN2RA2	PRGA
SCN2RC2	PRGA
SCN2RP1	PRG
SEQCHK	PRG

Table 8. Data Area Directory
(Part 12 of 14)

NAME	DATA AREA
SIGN	PRG(R)
SLFORM	PRG
SLOTSIZE	VFILTAB
SMASK1	NUC
SMASK3	NUC
SPANSWCH	PRG(R)
SQR2	VARCON
SRCCR	PRG
SRCLIN	PRG
SRCPTR	PRG
SREQST	PRG(C)
SSLFORM	PRG(C)
STATESW	PRG(C)
STATTAB	PRG
STCSV	PRGA
STCTSV2	PRG
STDCMP	PRGA
STKERR	OBJAREA
STMBRAD	STMTABLE
STMCOUNT	STMTABLE
STMEOT	STMTABLE
STMFLAGS	STMTABLE
STMFREQ	STMTABLE
STMLINNO	STMTABLE
STMPUID	STMTABLE
STMSCLAD	STMTABLE
STORE	ESPACE
STPRS	PRG
STRLNG	PRG(R)
STRSW	PRGA
SUBSCR2	OBJAREA
SUBSCR3	OBJAREA
SUBSERR	OBJAREA
SVCINST	PRG
SVOBJRG1	LRG
SVOBJRG4	PRG
SVRP4	PRG
SVSRCP	PRG
SWITCH	PRGA
SWPFLG	PRG
SYNTYPE	PRG(C)
THISCI#	VFILTAB
THISREC#	VFILTAB
THISUNIT	VFILTAB
TMBUF	PRG
TOUTSW	PRG(R)
TYPCODE	PRG(C)
UFUNARG	UFUN
UFUNDISP	UFUN
UFUNPTR	PRG(R)
UFUNWDSP	UFUN
UFUNWORK	UFUN
USCCW	PRG
USEPARM	PRG
UTTLOC	PRG
VACBA	VFILTAB
VACBLEN	VFILTAB
VACSLAST	VFILTAB
VALRP2	PRG
VALSW	PRG(C)
VAL4RA1	PRG(C)

Table 8. Data Area Directory (Part 13 of 14)

NAME	DATA AREA
VARCBASE	PRG(C)
VAREA	VFILTAB
VAREALN	VFILTAB
VAREALNP	VFILTAB
VAREALNR	VFILTAB
VAREALNW	VFILTAB
VAREAMNL	VFILTAB
VAREAMXL	VFILTAB
VARENDA	VFILTAB
VARPTRS	PRGA
VCEXCEPT	VFILTAB
VCKEYA	VFILTAB
VCKEYLN	VFILTAB
VCLEARA	VFILTAB
VCLEARLN	VFILTAB
VCOND	VFILTAB
VCURPOS	ESPACE
VDATA	ESPACE
VDATASUB	ESPACE
VDEFLT	ESPACE
VDINDEX	ESPACE
VDL	ESPACE
VDLSUB	ESPACE
VDTYPE	ESPACE
VERROR	VFILTAB
VEXITDSP	VFILTAB
VEXLLEN	VFILTAB
VEXLSTA	PRG(R)
VFILENO	VFILTAB
VFILNAME	VFILTAB
VFIRSTFM	ESPACE
VFLAREA	PRG(R)
VFMODE	VFILTAB
VFORG	VFILTAB
VFORMA	ESPACE
VFPROC	ESPACE
VFSPECA	ESPACE
VFTYPE	VFILTAB
VKEYED	VFILTAB
VKEYLEN	VFILTAB
VKEYLN	VFILTAB
VLAST	ESPACE
VLASTREG	VFILTAB
VLCOND	VFILTAB
VLDIFF	ESPACE
VLEXCEPT	VFILTAB
VLKEYA	VFILTAB
VLKEYED	VFILTAB
VLKEYLN	VFILTAB
VLNAREA	VFILTAB
VNAMELN	VFILTAB
VNKEYA	ESPACE
VNO	ESPACE
VPLAY	PRG(R)
VRBACUR	VFILTAB
VRBAEND	VFILTAB
VRBALAST	VFILTAB
VREQST	VFILTAB
VRKP	VFILTAB
VRPLA	VFILTAB
VRPLLEN	VFILTAB
VRUNSAVE	ESPACE
VSBASCHK	NUC
VSBATTN	PRG
VSBID	PRG
VSBLIB	PRG

Table 8. Data Area Directory (Part 14 of 14)

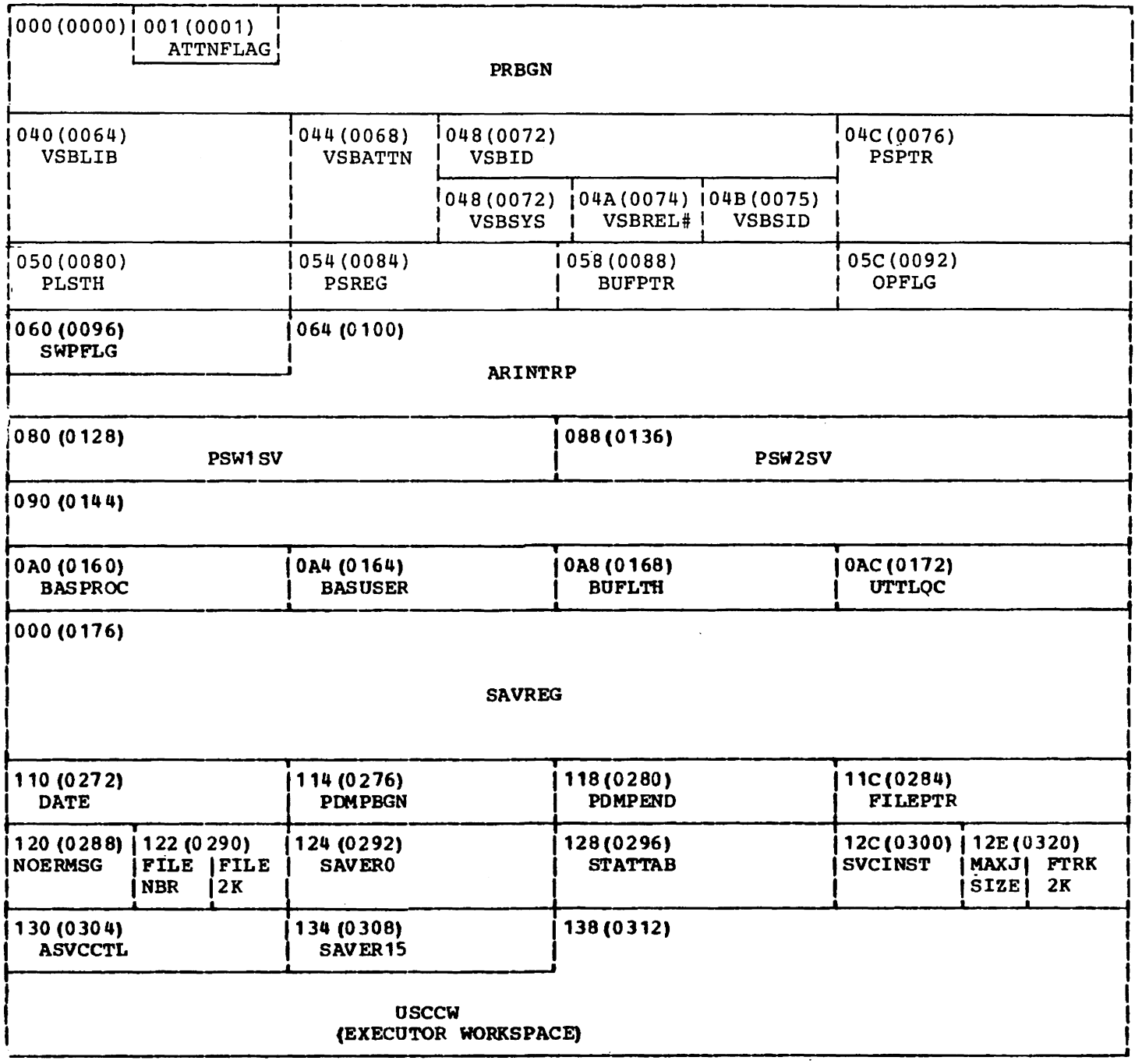
NAME	DATA AREA
VSBREL#	PRG
VSBSYS	PRG
VTCODE	ESPACE
VTITER	ESPACE
VTL	ESPACE
VTLSUB	ESPACE
WBG	PRG(C)
WCATCT1	PRGA
WCATLTHR	PRGA
WCATRESL	PRGA
WCATSAV	PRGA
WCMP	PRGA
WCOMCT	PRGA
WCUR	PRG(C)
WEXEC	PRGA
WFLG	PRGA
WFLG2	PRGA
WFNTLEV	PRGA
WLTH	PRG(C)
WLTHSAV	PRG(C)
WPARLEV	PRGA
WSYM	PRGA
WWK	PRGA

DATA AREA FORMATS

PRG -- COMMUNICATIONS REGION (COMPILATION AND RUN-TIME)

PRG contains information for communicating within and between routines of the VS BASIC Processor. This information includes base addresses, dynamic storage areas that are used and shared by compiler and run-time routines. Areas of PRG that are used by the compiler and overlaid by the run-time routines are shown separately.

PRG -- Communications Region (Compilation and Run-time)



PRG -- Communications Region (Compilation and Run-time) (continued)

200 (0512) OPTIONS	204 (0516) MAXFILES	208 (0520) CPPLADDR	20C (0524) DEBUGTAB
210 (0528) DBUGNTRY	214 (0532) DEBUG Secondary Entry Points		22C (0556) DBUGFLGS | DBUGRSRV
DBUGRSRV	234 (0564) SAVEBASE		23C (0572) SAVESTOR
240 (0576) RECOGNIZ		248 (0584) ATTNREG (EXECUTOR WORKSPACE)	
350 (0848) RUNPARM or BUFFAHED			
450 (1104) PLINE			
550 (1360) TMBUF (TERMINAL BUFFER)			
750 (1872) SLFORM | SLBYTE	754 (1876) BSLFORM	758 (1880) LINWDTH	75C (1884) PARMPTR
760 (1888) PSMAX	764 (1892) PSMIN	768 (1896) ASPTR	76C (1950) ASLTH
770 (1904) ASMAX	774 (1908) AVLIN	778 (1912) IMEND	77C (1916) EXTEND
780 (1920) SEQCHK	784 (1924) NEEDBUF	788 (1928)	78C (1932) USEPARM
790 (1936) IBFPTR	794 (1940) FPHCFLAG | 795 (1941) SAVSWTCH	798 (1944)	79C (1948) ERMFLAG FCNMRK | 79E (1950) MSGCNT
7A0 (1952) IFTMP		7A8 (1960) FORTMP	
7B0 (1968) RNDTMP		7B8 (1976) MATMP1	
7C0 (1984) MATMP2		7C8 (1992) ERRSTART	7CC (1996) ERREND
7D0 (2000) ERR-SAVED	COMPABS (see PRG - COMPABS (Compile Only) or PRG - COMPABS (Run-time Only) for contents)		

PRG -- Communications Region (Compilation and Run-time) (continued)

COMPABS	A74(2676) PSSAV or BSPRG	A78(2680) BSVARCN1	A7C(2684) BSVARCN2
A80(2688) BSOBJ	A84(2692) BSOVFLW	A88(2696) BSUFUN	A8C(2700) BSEXTPTR
A90(2704) BSDAT	A94(2708) CURDAT	A98(2712) ENDDAT	A9C(2716) BSKEY
AA0(2720) CURKEY	AA4(2724) BSLINCHN	AA8(2728) BSLNPTRS	AAC(2732) BSVARPTR
AB0(2736) BSLINTB	AB4(2740) BSPARMSV	AB8(2744) CSTART or OBJPTR	ABC(2748) STPRS or LETTMP
STPRS or LETTMP			ACC(2764) BSSRC
AD0(2768) SRCPTR	AD4(2772) BSLINTAB	AD8(2776) BSPRGA	ADC(2780) SRCLIN
AE0(2784) SRCCR	AE4(2788) SVRP4	AE8(2792) SCN1RP1	
AF0(2796) SCN1RP1 or SCN1RP3	AF4(2800) SCN2RP1		
B00(2816) VALRP2	B04(2820) SVOBJRG1	B08(2824) SVOBJRG4	AFC(2828) DEFSAV
B10(2832) LINSVP		B18(2840) FMLASVP or CTLSV1	B1C(2844) FMLASVP, CTLSV1 or CTLSV2
FMLASVP		B28(2856) MATRPSV	B2C(2860) SVSRCP
B30(2864) RELWORK		B38(2868) STCTSV2	B3C(2872) ARRSVP
ARRSVP		B48(2884) AARJPR	
B50(2896) LOCBRANC	B54(2900) IFRTMPO	B58(2904) IFRTMP1	B5C(2908) IFRTMP2

Name	Description
PRBGN	ENTRY POINT FOR EXECUTION
ATTNFLAG	ATTENTION HANDLING FLAG Contents: X'FE' = NO ATTENTION PENDING; X'0E' = ATTENTION PENDING
VSBLIB	ADDRESS OF VSBASIC LIBRARY
VSBATTN	DISPLACEMENT FROM VSBLIB TO ATTENTION ROUTINE
VSBSYS	FLAG FOR SYSTEM CHARACTERISTICS Contents: X'01' = NO VSAM; X'02' = NO EXTENDED PRECISION; X'04' = A LIBRARY FUNCTION REQUIRES EXTENDED PRECISION
VSBSID	ENVIRONMENT IDENTIFICATION Contents: F'2' = TSO; F'3' = CMS; F'4' = OS BATCH; F'5' = DOS BATCH; F'6' = VSPC
VSBREL#	VSBASIC RELEASE NUMBER Contents: X'03' = RELEASE 3
PSPTR	DISPLACEMENT FROM PRBGM TO LIST OF VALUES THAT MUST BE UPDATED WHEN AREA IS RELOCATED
PSLTH	LENGTH OF LIST POINTED TO BY PSPTR
PSREG	BIT MAP OF REGISTERS Contents: 0 = ABSOLUTE VALUE; 1 = RELOCATABLE VALUE
BUFPTR	DISPLACEMENT FROM TMBUF TO NEXT AVAILABLE BYTE FOR OUTPUT
OPFLG	OUTPUT INHIBIT FLAG Contents: 0 = BUFFER MAY BE WRITTEN; 1 = BUFFER MAY NOT BE WRITTEN
ARINTRP	LOCATION TO WHICH CONTROL IS TRANSFERRED ON AN ARITHMETIC INTERRUPT
PSW1SV	NOT USED
PSW2SV	PSW WHEN AN ARITHMETIC INTERRUPT OCCURS
BASPROC	PROCESSOR BASE ADDRESS
BASUSER	USER PROGRAM BASE ADDRESS
BUFLTH	LENGTH OF TERMINAL I/O BUFFER
UTTLOC	USER TERMINAL TABLE BASE ADDRESS
SAVREG	REGISTERS WHEN PROGRAM IS SAVED
DATE	NOT USED
PDMPBGN	NOT USED
PDMPEND	NOT USED
FILEPTR	DISPLACEMENT TO TABLE CONTAINING POINTERS TO FILE CONTROL BLOCKS
NOERMSG	NUMBER OF ERROR MESSAGES ISSUED
FILENBR	LOGICAL FILE NUMBER TO BE READ OR WRITTEN
FILE2K	NOT USED
SAVER0	NOT USED
STATTAB	NOT USED
SVCINST	LINKAGE INSTRUCTION TO REQUEST EXECUTOR SERVICES
MAXJSIZE	NOT USED
FTRK2K	NOT USED
ASVCCTL	ADDRESS OF SERVICE CALL ENTRY POINT
SAVER15	SAVE AREA FOR REGISTER 15 DURING LINKAGE
USCCW	EXECUTOR WORK SPACE (SPACE RESERVED FOR USER TERMINAL TABLE)
OPTIONS	BIT SWITCHES FOR OPTIONS Contents: X'01' = TEST OPTION SET; X'02' = MESSAGES SHOULD INCLUDE MESSAGE ID; X'04' = DO NOT CONTINUE AFTER ATTENTION; X'08' = INVERTED PRINT EDIT

Name	Description
MAXFILES	NUMBER OF ACTIVE FILES ALLOWED
CPPLADDP	ADDRESS OF CPPL
DEBUGTAB	ADDRESS OF AREA IN WHICH TABLES FOR DEBUG ARE BUILT
DBUGNTRY	MAIN ENTRY POINT TO DEBUG
DBUGFLGS	FLAGS TO CONTROL DEBUG ACTIVITY
DBUGRSRV	RESERVED
SAVEBASE	SAVE AREA FOR EXECUTOR'S BASE ADDRESSES (REGISTERS 11 and 12)
SAVESTOR	SAVE AREA FOR WORKSPACE BASE ADDRESS (REGISTER 13)
RECOGNIZ	RECOGNITION FIELD
ATTNREG	ATTENTION REGISTERS
RUNPARM	CHAINING PARAMETER AREA
BUFFAHED	AREA FOR BUFFERED AHEAD TERMINAL INPUT
PLINE	INTERNAL PRINT BUFFER
SLFORM	INDICATES SHORT OR LONG PRECISION Contents: 0 = SHORT; 1 = LONG
BSLFORM	
LINWIDTH	USER SPECIFIED LINE WIDTH
PAPMPTP	DISPLACEMENT OF NEXT ENTRY IN EXCEPTION TABLE
PSMAX	MAXIMUM SIZE OF THE RELOCATABLE PROGRAM SAVE AREA
PSMIN	
ASPTR	POINTER TO ABSOLUTE SAVE AREA
ASLTH	SIZE OF THE ABSOLUTE SAVE AREA
ASMAX	MAXIMUM SIZE OF THE ABSOLUTE SAVE AREA
AVLIN	DISPLACEMENT TO NEXT LINPTRS OR LINCHN ENTRY ('4')
IMEND	LENGTH OF IMPTRS TABLE
EXTEND	LENGTH OF EXIT TABLE
SEQCHK	SOURCE SEQUENCE CHECK FIELD ('-1')
NEEDBUF	COUNTER FOR NUMBER OF BUFFERS FOR FILES ('0')
USEPARM	HOLD AREA FOR USE VARIABLE (PDDD)
IBFPTR	
SAVSWTCH	SAVE AREA FOR RETSW
FPHCTLAG	FLOATING POINT HARD CHANGE FLAG (X'00')
ERMFLAG	DUPLICATE ERROR MESSAGE FLAG
FCNMPK	LEVEL OF SUBROUTINE NEST
MSGCNT	MESSAGE EMITTED FLAG
IFTMP	
FORTMP	
RNDTMP	
MATMP1	
MATMP2	
PSSAV	
BSPRG	BASE ADDRESS OF PROGRAM AREA (PROGA)
BSVARCN1	BASE ADDRESS OF VARCON1
BSVARCN2	BASE ADDRESS OF VARCON2
BSOBJ	BASE ADDRESS OF THE OBJECT AREA
BSOVFLW	BASE ADDRESS OF THE OBJECT AREA OVERFLOW
BSUFUN	BASE ADDRESS OF THE UFUN TABLE
BSEXTPTR	BASE ADDRESS OF THE EXTPTRS TABLE
BSDAT	BASE ADDRESS OF THE DATA TABLE
CURDAT	CURRENT LOCATION IN DATA TABLE
ENDDAT	END OF THE DATA TABLE
BSKEY	BASE ADDRESS OF THE KEY TABLE
CURKEY	CURRENT LOCATION IN KEY TABLE
BSLINCHN	BASE ADDRESS OF THE LINCHN TABLE
BSLNPTRS	BASE ADDRESS OF THE LNPTRS TABLE
BSVARPTR	
BSLINTB	
BSPARMSV	
OBJPTR	NEXT AVAILABLE BYTE IN THE OBJECT AREA
STPRS	STATEMENT PROCESSOR WORK AREA

Name	Description
BSSRC	BASE ADDRESS OF THE SOURCE PROGRAM
SRCPTR	CURRECT LOCATION IN THE SOURCE PROGRAM
BSLINTAB	BASE ADDRESS OF THE LINTAB TABLE
BSPRGA	BASE ADDRESS OF THE PRGA AREA
SRCLIN	LOCATION OF LINE NUMBER FOR LTHS LINE
SRCCR	RESERVED
SVRP4	RESERVED
SCN1RP1	SAVE AREA FOR ICDJSCN1 REGISTERS (RP1-RP3)
SCN2RP1	SAVE AREA FOR ICDJSCN2 REGISTERS (RP1-RP3)
VALRP2	SAVE AREA FOR ICDJVAL REGISTER (RP2)
SVOBJRG1	
SVOBJRG4	
DEFSAV	
LINSVP	
FMLASVP	SAVE AREA FOR ICDJFMLA RELOCATABLE REGISTERS
MATRPSV	
SVSRCP	
RELWORK	RELOCATABLE WORK AREA
STSTSV2	SAVE AREA FOR ICDJCTL
ARRSVP	SAVEAREA FOR ICDJALOC
AADJPR	SAVE AREA FOR ICDJAADJ
LOCBRANC	LOCATION OF BRANCH AROUND INSTRUCTION IN DEF
IFRTMP0	IF PROCESSOR POINTERS
IFRTMP1	
IFRTMP2	

PRG - COMPABS (Compile Only)

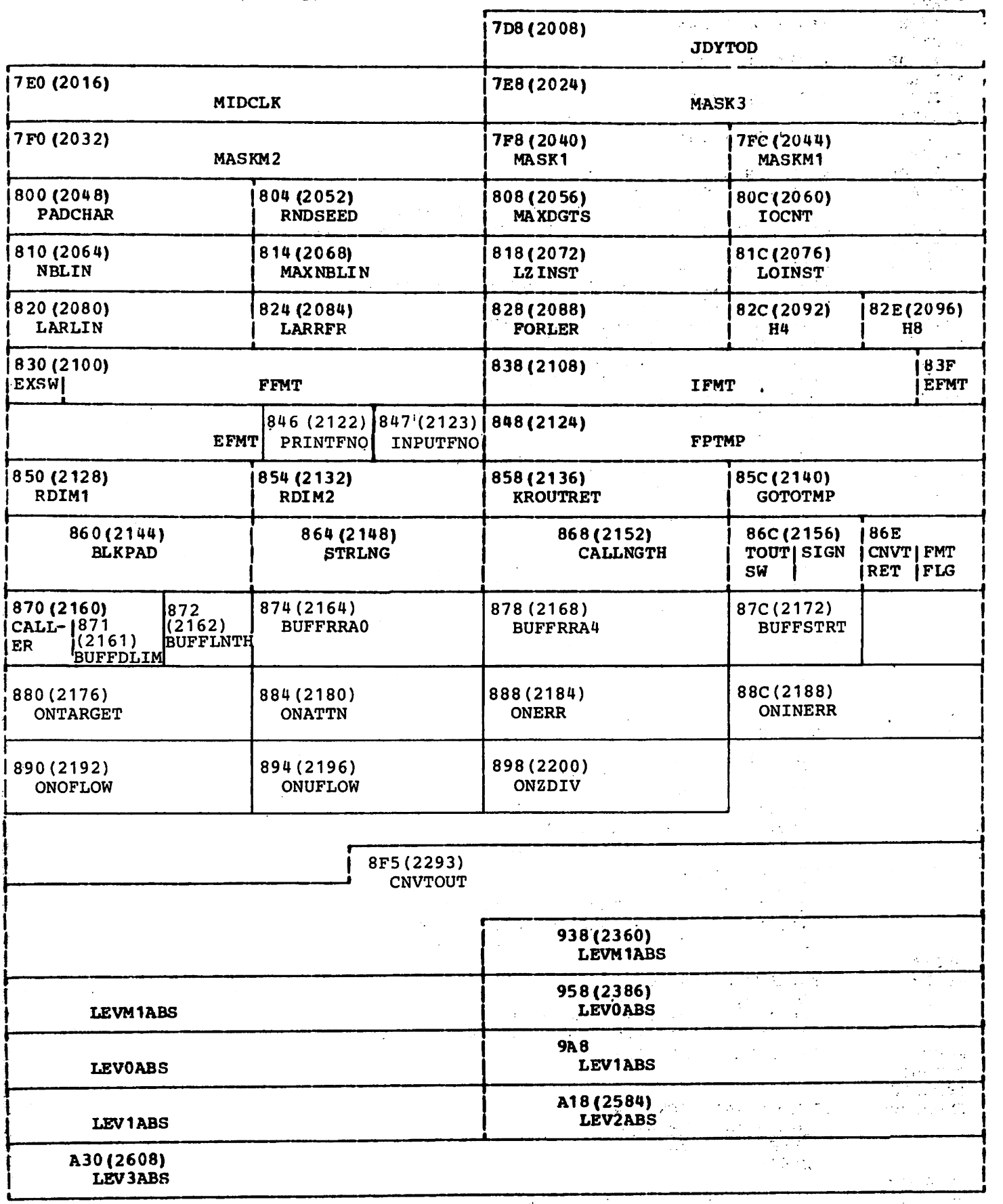
	7D4 (2004) DEFFORCT	7D8 (2008) CNOLINE	7DC (2012) ARRBYT
7E0 (2016) FSTDAT	7E4 (2020) FSTIM	7E8 (2024) FSTFRM	7EC (2028) FSTEXT
7F0 (2032) AVIM	7F4 (2036) AVEXT	7F8 (2040) WCUR	7FC (2044) WBG
800 (2048) WLTH	804 (2052) ERROCCUR	808 (2056) DEFNAME | 80A SYMTYPE	80C (2060) CNOERMSG |
810 (2064) WLTHSAV	814 (2068) DATAKEYS	818 (2072) STATE-SW	81C (2076) AVFOR
820 (2080) VARCBASE	824 (2084) ASLADD	828 (2088) ASLMASK	82C (2092) SSLFORM
830 (2096) CURBSREG	834 (2100) AVVAR	838 (2104) ENDVAR	83C (2108) ENDLIN
840 (2112) NOLINE	844 (2116) $SW | $AW | $DE | $MD $SU | $AU | $DD | $ME	848 (2120) $SD | $AD | $CD | $LD $SE | $BE | $CE | $LE	84C (2124) $STD | $SWI | $AWR | $DDR $STE | $SWR | $AUR | $DER
850 (2128) $MDR | $SDR | $ADR | $CDR $MER | $SER | $AER | $CER	854 (2132) $LDR | $HDR | $LCDR | $LTDR $LER | $HER | $LCER | $LTER	858 (2136) $LNDR | $LPDR $LNER | $LPER	85C (2140) VAL4RA1
860 (2144) CURCTL	864 (2148) CURDEF	868 (2152) CURLINE	86C (2156) PARMCNT
870 (2160) MATDABS			87C (2172) MATDRC1
880 (2176) MATDRC3	884 (2180) RDIMRC1	888 (2184) RDIMRC3	88C (2188) RDABS
RDABS		898 (2200) ABSWORK	
ABSWORK	8A4 (2212) ARRSVA		
ARRSVA		8B8 (2232) ARRSVC	
ARRSVC	8C4 (2244) AADJAR		
AADJAR	8D4 (2260) PARMTBL		
		938 (2360) LINSAV | LIN-SAV1 | LIN-SAV2	93C (2364) UFUNPTR | 93E (2366) RET-SW | VAL-SW
940 (2368) GEN-FLAG | QT-CHAR | 942 (2370) RE-DIM | MODE | 944 (2372) SRE-QST | TYP-CODE | 946 (2374) FXDPT-OVF		948 (2376) IMFRMCNT	94C (2380) HDRCDDBG
952 (2386) HDRCDSTD			

Name	Description
DEFFORCT	COUNTER OF 'FOR' LOOPS IN USER FUNCTION
CNOLINE	CURRENT NUMBER OF SOURCE STATEMENTS PROCESSED
ARRBYT	NEXT AVAILABLE BYTE IN CURRENT ARRAY BLOCK
FSTDAT	POINTER TO FIRST DATA STATEMENT ENTRY
FSTIM	POINTER TO FIRST IMAGE STATEMENT ENTRY
FSTFRM	POINTER TO FIRST FORM STATEMENT ENTRY
FSTEXT	POINTER TO FIRST EXIT STATEMENT ENTRY
AVIM	DISPLACEMENT TO NEXT AVAILABLE IMPTRS ENTRY
AVEXT	DISPLACEMENT TO NEXT AVAILABLE EXTPTRS ENTRY
WCUR	
WBG	
WLTH	
ERROCCUR	
DEFNAME	FUNCTION NAME (WITHOUT FN)
SYMTYPE	TYPE OF INSTRUCTION
CNOERMSG	
WLTHSAV	
DATAKEYS	NUMBER OF KEY TABLE ENTRIES
STATESW	STATEMENT PROCESSOR SWITCH

Contents	Meaning
X'80'	PROCESSING AN IF STATEMENT
X'40'	PROCESSING A MULTI-LINE FUNCTION
X'20'	SRC WINDOW REQUESTED IN ERROR MESSAGE
X'10'	TEST OPTION REQUESTED
X'08'	RETURN IN A DEF STATEMENT
X'04'	ARITHMETIC FUNCTION
X'02'	PROCESSING A MAT STATEMENT
X'01'	STATEMENT NUMBER IS A STATEMENT LABEL

AVFOR	DISPLACEMENT OF NEXT AVAILABLE END FOR ENTRY
VARCBASE	BASE ADDRESS OF VARCON AREA
ASLADD	
ASLMASK	
SSLFORM	
CURBSREG	CURRENT BASE REGISTER FOR THE VARCON AREA
AVVAR	DISPLACEMENT TO NEXT AVAILABLE VARCON ENTRY
ENDVAR	LENGTH OF THE VARCON AREA
ENDLIN	LENGTH OF THE LINPTRS OR LINCHN TABLES
NOLINE	NUMBER OF SOURCE LINES IN PROGRAM
$SU/$SW	X'7F' OR X'6F'
$AU/$AW	X'7E' OR X'6E'
$DE/$DD	X'7D' OR X'6D'
$ME/$MD	X'7C' OR X'6C'
$SE/$SD	X'7B' OR X'6B'
$AE/$AD	X'7A' OR X'6A'
$CE/$CD	X'79' OR X'69'
$LE/$LD	X'78' OR X'68'
$STE	
$STD	X'70' OR X'60'
$SUR/	
$SWR	X'3F' OR X'2F'
$AUR/	
$AWR	X'3E' OR X'2E'
$DER/	
$DDR	X'3D' OR X'2D'
$MER/	
$MDR	X'3C' OR X'2C'
$SER/	
$SDR	X'3B' OR X'2B'
$AER/	
$ADR	X'3A' OR X'2A'
$CER/	
$CDR	X'39' OR X'29'
$LER/	
$LDR	X'38' OR X'28'

Name	Description
$HER/	
$HDR	X'34' OR X'24'
$LCER/	
$LCDR	X'33' OR X'23'
$LTER/	
$LTDR	X'32' OR X'22'
$LNER/	
$LNDR	X'31' OR X'21'
$LPER/	
$LPDR	X'30' OR X'20'
VAL4RA1	SAVE AREA FOR ICDJVAL4
CURCTL	DISPLACEMENT OF CURRENT CONTROL ROUTINE
CURDEF	DISPLACEMENT OF CURRENT DEFERRED STATEMENT ENTRY
CURLINE	
PARMCNT	PARAMETER COUNTER
MATDABS	SAVE AREA FOR ICDJMATD ABSOLUTE REGISTER 1
MATDRC1	SAVE AREA FOR ICDJMATD REGISTER
MATDRC3	SAVE AREA FOR ICDJMATD REGISTER
RDIMRC1	SAVE AREA FOR ICDJBDIM REGISTER
RDIMRC3	SAVE AREA FOR ICDJPDIM REGISTER
RDABS	SAVE AREA FOR ICDJPDIM REGISTERS
ABSWORK	WORKSPACE TO CHANGE B REGISTERS TO A REGISTERS
ARRSVA	SAVE AREA FOR ICDJALOC ABSOLUTE REGISTER
APRSVC	SAVE AREA FOR ICDJALOC
AADJAR	SAVE AREA FOR ICDJAAPJ ABSOLUTE REGISTERS
UFUNPTR	DISPLACEMENT OF DEF ENTRY IN UFUN TABLE
RETSW	RETURN STATEMENT SWITCH
	Contents: Meaning
	X'00': RETURN TO ICDJFMLA
	X'01': RETURN TO DIM STATEMENT PROCESSING
	X'02': RETURN TO DEF STATEMENT PROCESSING
	X'04': RETURN TO LET, NEXT, FOR, USE, OR FORM STATEMENT PROCESSING
	X'20': RETURN TO INPUT OR READ STATEMENT PROCESSING
	X'08': RETURN TO GET STATEMENT PROCESSING
	X'10': RETURN TO OPEN/CLOSE/RESET
	X'28': REDIMENSIONING CORRECT; 'CALL ID' NEEDED
	X'38': REDIMENSIONING CORRECT
	X'FF': REDIMENSIONING NOT PERMITTED
VALSW	ENTRY INDICATOR USED IN ICDJVAL
	Contents: Meaning
	X'00': ENTRY AT ICDJVAL1
	X'01': ENTRY AT ICDJVAL2
	X'03': ENTRY AT ICDJVAL3
	X'07': ENTRY AT ICDJVAL4
	X'80': ENTRY AT ICDJVAL5
GENFLAG	GENERAL FLAG BYTE
	Contents: Meaning
	X'01': SPACE USED FOR VARCON2
	X'02': SPACE USED FOR OBJECT CODE
	X'40': UNRECOVERABLE ERROR FLAG
QTCHAR	
REDIM	REDIMENSION SWITCH
	Contents: Meaning
	X'01': REDIMENSION OCCURRED
	X'00': REDIMENSIONING DID NOT OCCUR
MODE	'IF' PROCESSING MODE
SREQST	
TYPCODE	
FXDPTOVF	FIXED POINT OVERFLOW
	Contents: Meaning
	X'FF': ON
	X'00': OFF
IMFRMCNT	
HDRCDDBG	STATEMENT HEADER CODE FOR DEBUG
HDRCDSTD	NORMAL STATEMENT HEADER CODE



PRG - COMPABS (Run-time Only) (continued)

A40(2624) ERSVRET	A44(2628) ERSVERRL	A48(2632) ERSVRET2	A4C(2636) ERABS
ERABS		A58(2648) RDECNO1	
A60(2656) RDECNO2		A68(2664) VPLAY	A6C(2668) VEXLSTA
A70(2672) VFLAREA			

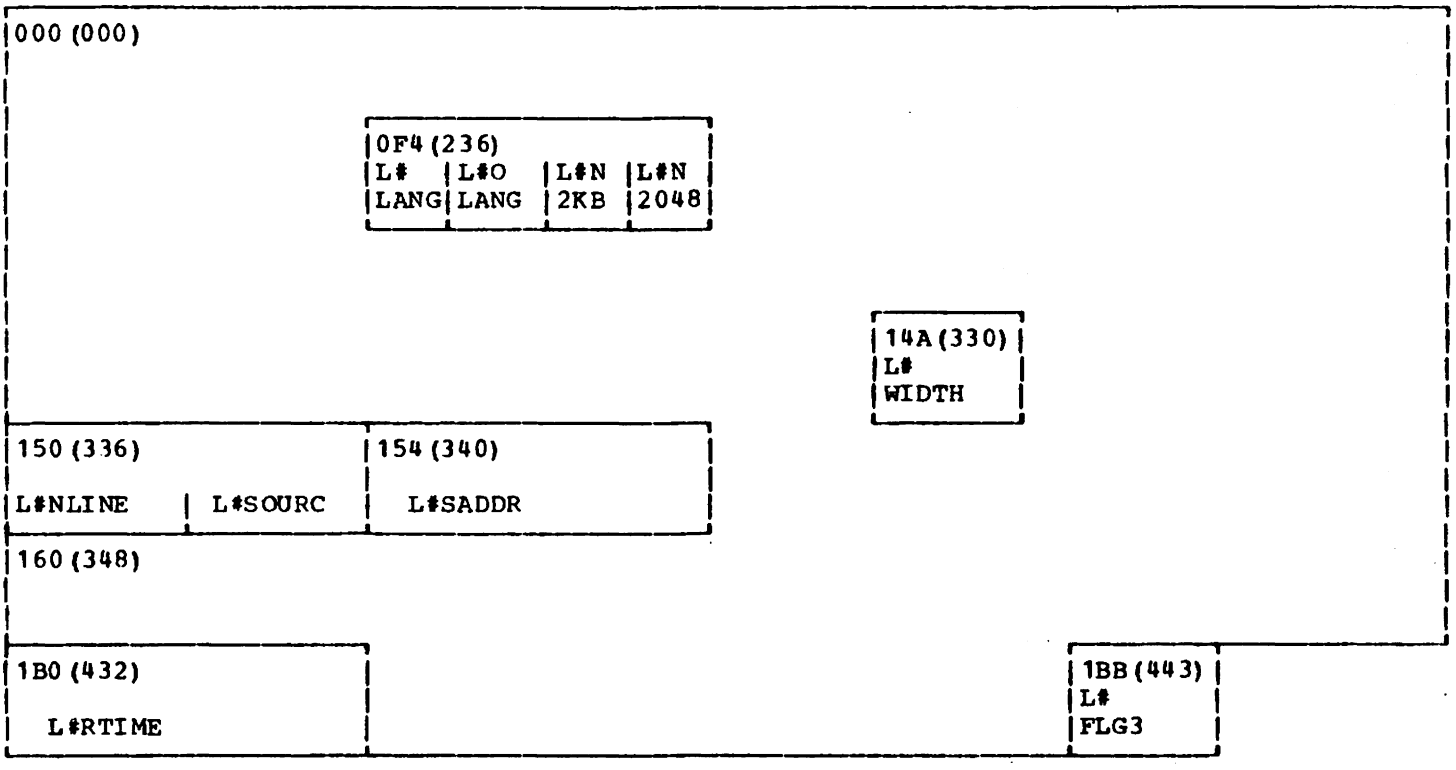
Name	Description
JDYTOD	TODAY'S JULIAN DATE
MIDCLK	PREVIOUS MIDNIGHT CLOCK
MASK3	X'4E00000000000000'
MASKM2	X'4E00000000000001'
MASK1	X'46000000'
MASKM1	X'46000001'
PADCHAR	X'40000000'
RNDSEED	HOLD AREA FOR A PSEUDO-RANDOM SEED
MAXDGTS	MAXIMUM NUMBER OF DIGITS Contents: 6 — SHORT PRECISION 15 — LONG PRECISION
IOCNT	NUMBER OF ARGUMENTS PUT OUT DURING I/O OPERATION
NBLIN	NUMBER OF CHARACTERS IN PRINT LINE
MAXNBLIN	HIGHEST POSITION USED IN PRINT LINE
L2INST	LOAD SINGLE INTO FLOAT REGISTER 2
L0INST	LOAD SINGLE INTO FLOAT REGISTER 0
LARLIN	LOAD ADDRESS INTO RELOCATABLE REGISTER RLIN
LARRFR	LOAD ADDRESS INTO RELOCATABLE REGISTER RRFR
FORLER	
H4	H'4'
H8	H'8'
EXSW	INDICATOR FOR EXECUTION STATUS IOSW — X'01' — I/O STATEMENT BEING PROCESSED LINERRSW — X'02' — ERROR MESSAGE TO SUPPRESS LINE NUMBER MATHLEV — X'04' — LEVEL INDICATOR FOR MATH ROUTINES
FFMT	SINGLE PRECISION F-FORMAT
IFMT	SINGLE PRECISION I-FORMAT
EFMT	SINGLE PRECISION E-FORMAT
PRINTFNO	PRINT TO FILE NUMBER
INPUTFNO	INPUT FROM FILE NUMBER
FPTMP	ARITHMETIC INTERRUPT FP REGISTER TEST
RDIM1	REDIMENSIONING TEMPORARY FOR DIMENSION 1
RDIM2	REDIMENSIONING TEMPORARY FOR DIMENSION 2
KROUTRET	RUN-TIME RETURN ADDRESS FOLLOWING EVALUATION OF ARG.
GOTOTMP	WORK AREA FOR COMPUTED GOTO OR GOSUB
BLKPAD	BLANK PADDING INFORMATION
STRLNG	STR LENGTH, INFORMATION FOR ICDKPLIN
CALLNGTH	CHAR STRING LENGTH
TOUTSW	OUTPUT INDICATOR X'01' — OUTPUT PRINT LINE AND CR TO TMBUF X'04' — OUTPUT PRINT LINE, CR, AND ? TO TMBUF X'08' — OUTPUT PRINT LINE, CR, AND ?? TO TMBUF
SIGN	CNVT VALUE INDICATOR X'00' — NUMBER IS POSITIVE OR ZERO X'01' — NUMBER IS NEGATIVE

Name	Description
CNVTRET	RETURN INDICATOR Contents: Meaning X'01': PRINT OUTPUT X'02': STREAM FILE OUTPUT X'04': VSAM OUTPUT X'08': CALL FROM DEBUG X'10': CALL FROM CHR FUNCTION
FMTFLG	FORMAT TYPE INDICATOR Contents: Meaning X'00': UNFORMATTED PRINT X'01': FORM-TYPE FORMAT X'02': IMAGE-TYPE FORMAT
CALLER	INPUT FLAG Contents: Meaning X'00': SKIP OVER EMBEDDED BLANKS X'01': EMBEDDED BLANKS TERMINATE SCAN X'02': END ADDRESS SUPPLIED X'04': CALLER IS THE GET ROUTINE X'10': CALLER IS THE INPUT ROUTINE X'20': CALLER IS THE DEBUG ROUTINE X'40':
BUFFWRK	BUFFERED AHEAD WORK AREAS
BUFFDLIM	SINGLE/DOUBLE QUOTE FLAG
BUFFLNTH	LENGTH OF INPUT LINE
BUFFRRA0	RRA0 SAVE AREA
BUFFRRA4	RRA4 SAVE AREA
BUFFSTRT	OFFSET TO NEXT INPUT LINE
ONTARGET	ON CONDITION TO BE USED
ONATTN	CELL FOR ON ATTN
ONERR	CELL FOR ON ERR
ONINERR	CELL FOR ON INERR
ONOFLOW	CELL FOR ON OFLOW
ONUFLOW	CELL FOR ON UFLOW
ONZDIV	CELL FOR ON ZDIV
CNVTOUT	OUTPUT BUFFER FOR CONVERSION ROUTINE
LEVM1ABS	WORK AREA FOR LEVEL -1 ROUTINES
LEV0ABS	WORK AREA FOR LEVEL 0 ROUTINES
LEV1ABS	WORK AREA FOR LEVEL 1 ROUTINES
LEV2ABS	WORK AREA FOR LEVEL 2 ROUTINES
LEV3ABS	WORK AREA FOR LEVEL 3 ROUTINES
ERSVRET	RETURN REGISTER SAVE AREA FOR ERROR PROCESSING
ESVERRL	SAVE AREA FOR ERROR LINE
ERSVRET2	
ERABS	REGISTER WORK AREA FOR ERROR PROCESSING
RDECNO1	CONVERSION WORK AREA
RDECNO2	CONVERSION WORK AREA
VPLAY	VSAM
VEXLSTA	VSAM
VFLAREA	VSAM

Z#UTT - USER TERMINAL TABLE

The User Terminal Table contains system/type information about the current VS BASIC user.

Z#UTT (User Terminal Table - VSPC, TSO, and CMS only)



Z#UTT

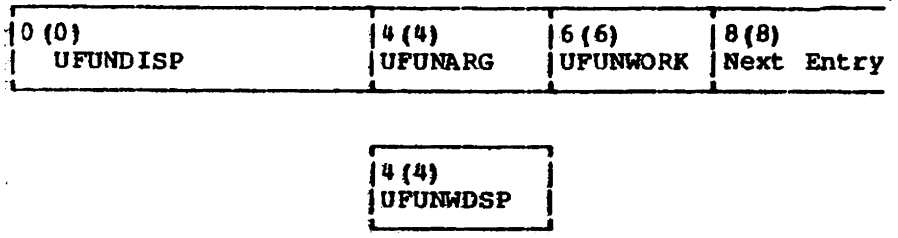
Name	Description
L#LANG	Language processor number
L#OLANG	Old language processor number
L#N2KB	New core size as requested by compiler
L#N2048	Number of 2K blocks of core assigned
L#WIDTH	Line width of terminal in characters
L#NLINE	Number of lines of sorted source
L#SOURC	Length of sorted source in bytes
L#SADDR	Displacement from base of user area to source code
L#RTIME	Current problem running time in microseconds
L#FLG3	General flag byte 3 meaning by bits

Contents	Meaning
X'02'	Long-form arithmetic

UFUN - USER FUNCTION TABLE

The User Function Table contains information required to process user functions in a VS BASIC program. There is one entry in the table for each user function that is defined in the program.

UFUN (Format of User Function Table Element)



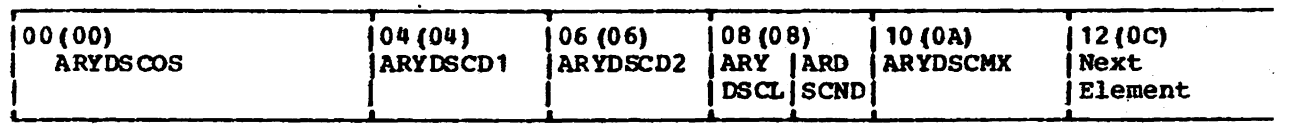
UFUN Table Entries

Name	Description
UFUNDISP	Displacement to function expansion
UFUNARG	Number of arguments
UFUNWORK	Length of WRKTMP (Compile time)
UFUNWDSP	Displacement to WRKTMP (Run time)

ARYDSC - ARRAY DESCRIPTION TABLE

The Array Description Table contains information about the arrays that are used in a VS BASIC program. Each array is described in one 12-byte element. A particular element can be found by using the appropriate displacement in the array pointers table (ARRPTRS).

ARYDSC (Format of Array Table Element)



ARYDSC Table Entries

Name	Description
ARYDSCOS	Offset from the beginning of the array area
ARYDSCD1	Current maximum value of the first subscript
ARYDSCD2	Current maximum value of the second subscript
ARYDSCL	Length in bytes of each element
ARYDSCND	Number of dimensions
ARYDSCMX	Maximum number of elements in array

ARRPTRS - ARRAY POINTERS TABLE

The Array Pointers Table is pre-allocated and contains 58 half-words. Each half-word is assigned to one possible array name and contains the displacement to the corresponding description of the array in the Array Description Table.

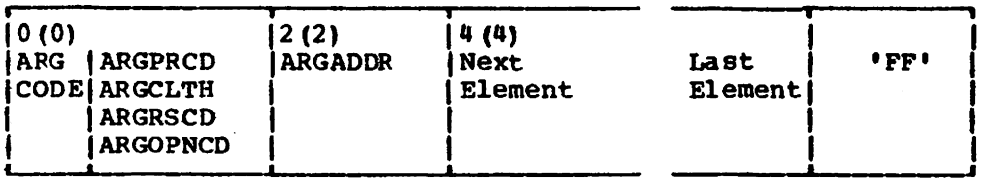
ARRPTRS (Array Pointers Table)

00 (00) A	02 (02) B	04 (04) C	06 (06) D	08 (08) E	0A (10) F	0C (12) G	0E (14) H
10 (16) I	12 (18) J	14 (20) K	16 (22) L	18 (24) M	1A (26) N	1C (28) O	1E (30) P
20 (32) Q	22 (34) R	24 (36) S	26 (38) T	28 (40) U	ZA (42) V	2C (44) W	2E (46) X
30 (48) Y	32 (50) Z	34 (52) @	36 (54) #	38 (56) $	3A (58) A$	3C (60) B$	3E (62) C$
40 (64) D$	42 (66) E$	44 (68) F$	46 (70) G$	48 (72) H$	4A (74) I$	4C (76) J$	4E (78) K$
50 (80) L$	52 (82) M$	54 (84) N$	56 (86) O$	58 (88) P$	5A (90) Q$	5C (92) R$	5E (94) S$
60 (96) T$	62 (98) U$	64 (100) V$	66 (102) W$	68 (104) X$	6A (106) Y$	6C (108) Z$	6E (110) @$
70 (112) #$	72 (114) $$	74 (116)					

ARGENTRY - ARGUMENT TABLE

The Argument Table describes in a pre-set format any arguments that may be required in a VS BASIC program. The arguments are generated in-line for each statement that required them. When the arguments are passed to a routine in the VS BASIC Processor, register 14 points to ARGENTRY.

ARGENTRY (Format of Argument Table Element)



ARGUMENT Table Entries

Name	Description
ARGCODE	Identifier type

Contents	Meaning
X'00'	Print null argument
X'01'	Expression
X'02'	Array
X'03'	Function
X'04'	Array element
X'08'	Character type
X'18'	String pseudo-variable
X'20'	Constant
X'30'	Condition keyword
X'80'	Exit keyword
X'40'	Object code return indicator
X'FF'	End of argument list

ARGPRCD	Print codes
ARGCLTH	Length for character strings
ARGRSCD	Reset end codes
ARGOPNCD	Open parameter codes
ARGADDR	BDDD address of identifier location

LINTAB - LINE TABLE

The Line Table contains offsets for each element in the Line Pointers Table and the Line Chain Table. The offsets in each table are the same for corresponding elements. These offsets are used by the run-time error processor to determine the location of the object code for each statement in a VS BASIC program. In addition, it contains displacements used by the compiler to locate deferred statements.

LINTAB (Format of Line Table Element)

0 (0)	2 (2)	4 (4)	6 (6)	8 (8)
LINCHN1	LINCHN2	LINSRCE	LINFNO	Next Element

LINTAB Table Entries

Name	Description
LINCHN1	Displacement into LINPTRS/LINCHN tables for next entry in hash chain
LINCHN2	Displacement into LINTAB for next entry in chain of deferred statements
LINSRCE	Displacement from beginning of source to first character after line number
LINFNO	Sequence number of user function in which statement appears

LINPTRS - LINE POINTERS TABLE

The Line Pointers Table contains the location, relative to the base address of the object code (BSOBJ), of the object code for each statement in the VS BASIC program. The high order portion of the line number is shown with each address. The low order portion of the line number is contained in the Line Chain Table (LINCHN.)

LINPTRS (Format of Line Pointers Table Element)

0 (0)	1 (1)	4 (4)
LINUM1	LINOBJ	Next Element

LINPTRS Table Entries

Name	Description
LINUM1	High order byte of binary line number
LINOBJ	Displacement from OBJAREA base address to code expansion

LINCHN - LINE CHAIN TABLE

The Line Chain Table contains, at the same offset as in the Line Pointers Table, the low order portion of the line number for each statement listed in the Line Pointers Table. In addition, this table also contains the offset to the next element in the table.

LINCHN (Format of Line Chain Table Element)

0 (0)	2 (2)	4 (4)
LINCHN3	LINUM2	Next Element

LINCHN Table Entries

Name	Description
LINCHN3	Displacement into LINPTRS/LINCHN tables of entry for next line number in numeric sequence
LINUM2	Low order bytes of the binary line number

INFOTAB - INFORMATION TABLE

The Information Table contains information about each intrinsic function that is referred to in the current VS BASIC program. This table is the only component of the compiler module ICDJINFO.

INFOTAB (Format of Information Table Element)

0 (0)	3 (3)	4 (4)	5 (5)	6 (6)	7 (7)	8 (8)
INFONAME	INFO RESV	INFO CD	INFO FLG	INFO NXT	INFO BIF	Next Element

INFOTAB Table

Name	Description
INFONAME	Function name in EBCDIC
INFORESV	Reserved
INFOCD	Function type Contents: X'00' Meaning: Numeric type Contents: X'08' Meaning: Character type
INFOFLG	Function description flags Contents: X'20' Meaning: BASIC intrinsic function Contents: X'10' Meaning: Arguments are required Contents: X'08' Meaning: Processed by generated code Contents: X'04' Meaning: No arguments Contents: X'02' Meaning: Numeric, single argument mathematical function Contents: X'00' Meaning: I/O condition keyword
INFONXT	Sequence number of the next entry in the hash chain
INFOBIF	BIFTAB/generated code entry number

EXTPTRS - EXIT POINTERS TABLE

The Exit Pointers Table contains pointers to the appropriate entry in the Condition Table (CNDTBL) for each EXIT statement in the program. The File Table (FILTAB) and the VSAM File Table (VFILTAB) contain index values into the Exit Pointers Table.

EXTPTRS (Format of Exit Table Element)

0 (0)	4 (4)
EXTDISP	Next Element

EXTPTR Table Entries

Name	Description
EXTDISP	Displacement from ROBJ to CNDTBL entry

CNDTBL - CONDITION TABLE

A Condition Table is built for each EXIT statement and group of I/O conditions, which are treated as if they were an EXIT statement, in the current VS BASIC program. Each entry in this table is either 0 or the address of the routine that handles the required error type. The EXTPTRS table contains pointers to CNDTBL.

CNDTBL (Format of Condition Table Element)

00 (00)	04 (04)	08 (08)	0C (12)
IOERRENT	EOFENT	CONVENT	NOKEYENT
10 (16)	14 (20)	18 (24)	1C (28)
DUPKYENT	NORECENT	DUPRCENT	NEXT ELEMENT

CNDTBL Table Entries

Name	Description
IOERRENT	Displacement to IOERR entry
EOFENT	Displacement to EOF entry
CONVENT	Displacement to CONV entry
NOKEYENT	Displacement to NOKEY entry
DUPKYENT	Displacement to DUPKEY entry
NORECENT	Displacement to NOREC entry
DUPRCENT	Displacement to DUPREC entry

ESPACE

ESPACE defines the temporary file area EFILTEMP, which holds information about a file before a file table has been allocated for it. The second part of ESPACE is used by library and executor routines as storage for save areas, additional file information, and data-list processing. BSEFIL contains a pointer to ESPACE.

000 (0000) ENAMELN	002 (0002)				
	EPILENM				
	014 (0020) ETYPE	EOPN-MODE	016 (0022) EFILE-NO	EV-EXCEPT	018 (0024) EPORG	EFREE	01A (0026) EXITDSP	01C (0028) E.ETNCOD
020 (0032) EACBA		024 (0036) EAREA			028 (0040) EAREALN			02C (0044) ERKP
030 (0048) ECKEYA		034 (0052) ELKEYA			038 (0056) ERPLA			03C (0060) EAREALNR
040 (0064) EAREALNW		044 (0068) EAREALNP			048 (0072) EARENDA			04C (0076) ERBACUR
050 (0080) EKEYLN	052 (0082) EREQ							
	UNUSED							
	074 (0116) ECLRA				078 (0120) ECLRLN			07C (0124) ERNL
080 (0128) ERXL		084 (0132)						
	ERECORD (124 bytes)							
100 (0256) EKEYED	102 (0258) EOPEN	EOPN-STAT	E-REQST	104 (0260) ER-MODE	EPLIN-CR	106 (0262) EKEY-INCR	EACTSEQ (24 bytes)			
120 (0288) DTEMP							128 (0296) ECURACT	12A (0298)	12C (0300)	
	VRUNSAVE (64 bytes)									
									16C (0362) VDINDEX	
170 (0368) VDATA				174 (0372) VD-TYPE	VDL		178 (0376) VNO		18C (0380) VTITER	
180 (0384) CONVER SW	182 (0386) FILL-CHAR	VT-CODE	VTL	184 (0388) VDLSUB			188 (0392) VTLSUB		18C (0396) NMOVE	
190 (0400) VFORMA				194 (0404) VFSPECA			198 (0408) V-DEFLT	19A (0410) VFIRST PM	VLDIFF	19C (0402) VNKEYA
1A0 (0416) V-LAST	VP-PROC			1A4 (0420) VCURPOS			1A8 (0424) LA-CODE		1AA (0426) ADDRESS	1AC (0428) STORE
1B0 (0432) MAXADDR				1B4 (0436) VDATASUB			1B8 (0440) SAVRLOC			1BC (0444) SAVFORMA
1C0 (0448) DATAWORD				1C4 (0452) RETURN			1C8 (0456) LOCWORD			1CC (0460) FORM-INCR	ERR-CODE	1CE (0462) ECLS-OPEN	HI-WRITE

Name	Description
ENAMELN	Length of DDNAME
EFILENM	DDNAME
ETYPE	File type
EOPNMODE	Mode for opening file; N/A for open request

Value	Meaning
X'00'	If not open, don't open (REREAD, CLOSE, END)
X'01'	If not open, open for input (READ, RESET)
X'02'	If not open, open for output (WRITE)
X'03'	If not open, open for ALL (REWITE, DELETE)

EFILENO	File number
EVEXCEPT	Error code
EFORG	File organization
EFREE	Reserved
EXITDSP	Exit pointer
ERETNCOD	Return code
EACBA	Reserved
EAREA	Reserved
EAREALN	Reserved
ERKP	Reserved
ECKEYA	Reserved
ELKEYA	Reserved
ERPLA	Reserved
EAREALNR	Reserved
EAREALNW	Reserved
EAREALNP	Reserved
ERENDA	Reserved
ERBACUR	Reserved
EKEYLN	Reserved
EREQ	Run-time request
ECLRA	Reserved
ECLRLN	Reserved
EMNL	Reserved
EMXL	Reserved
ERECORD	Reserved
EKEYED	Indicates presence/absence of key clause
EOPEN	Indicates whether open is legal/illegal
EOPNSTAT	Indicates file status (open/closed)
EPARAMS	Parameters of current request
EREQST	Request code

Value	Request	Value	Request
X'01'	Read	X'07'	Open
X'02'	Write	X'08'	Read
X'03'	Rewrite	X'09'	Close
X'05'	Delete	X'10'	End
X'06'	Reset		

ERMODE Indicates the mode in which the request is invalid. This flag is compared with the open mode; if equal, the error condition is handled.

Value	Meaning
X'00'	Request is always valid
X'01'	Invalid for write (file is open for input)
X'02'	Invalid for read, reread (file is open for output)
X'0C'	Invalid for rewrite, delete since these requests require MODE=ALL

EPLINCR Length of request parameter list (excluding data)

Value	Request
X'04'	Open, close, end
X'08'	Reset
X'10'	Read, reread, write, rewrite, delete

EKEYINCR Position of key in argument list

Value	Request
X'00'	Open, close
X'04'	Reset
X'0C'	Read, reread, write, rewrite, delete

EACTSEQ	Sequence of steps in module ICDKVIOR to handle request
DTEMP	Temporary area
ECURACT	Current step
VRUNSAVE	Run-time register save area
VDINDEX	Data count field

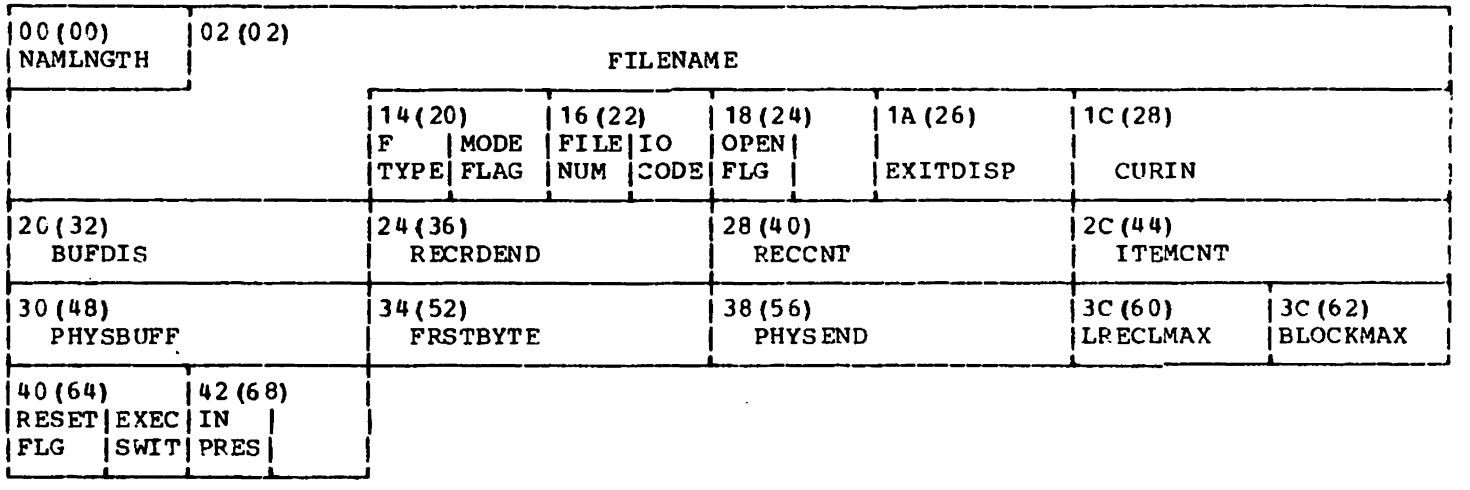
Name	Description
VDATA	Address of data item
VDTYPE	Type of data item
VDL	Length of data item
VNO	Number of elements in data item
VTITER	Value of FORM iteration
CONVERSW	Indicates whether data conversion is required; used as save area for EOPNMOD during file scan
FILLCHAR	Padding character (0, if numeric data; blank, if character data)
VTCODE	FORM code
VTL	Length of FORM data
VDLSUB	Data length
VTLSUB	FORM length
NMOVE	Number of items to move
VFORMA	Address of current FORM
VFSPECA	Address of FORM start
VDEFLT	True if no FORM present
VFIRSTFM	Reserved
VLDIFF	Reserved
VNKEYA	Address of null key
VLAST	On if all data sent
VFPROC	Reserved
VCURPOS	Current buffer position
LADDRESS	Variable load address instruction
LACODE	Instruction code
ADDRESS	Instruction address
STORE	Variable STE instruction
MAXADDR	Highest buffer position reached
VDATASUB	Data address hold area
SAVRLOC	Relocation save area
SAVFORMA	FORM address save area
DATAWORD	Data save area
RETURN	Return address save area
LOCWORD	Location save area
FORMINCR	FORM item length
ERRCODE	Error code
ECLSOPEN	Indicates whether file's initial load status is to be altered
HIWRITE	True if current key is high

FILETAB - FILE TABLE (STREAM I/O) AND

VFILTAB - VSAM FILE TABLE (RECORD I/O)

These run-time tables are obtained dynamically for each open file referred to in the current VS BASIC program. They contain information about the current status of the files they describe. The beginning portion of both FILTAB and VFILTAB are formatted in the same way for ease of processing without regard to the type of I/O (stream or record).

FILTAB (File Table for Stream and Record Files)



FILTAB Table Entries (Stream and Record Files)

Name	Description
NAMLNGTH	Length of file name
FILENAME	DDname of file
FTYPE	Type of file

Contents	Meaning
X'00'	Stream
X'01'	Record
X'80'	Undefined file
X'40'	'PRINT TO/INPUT FROM' file

MODEFLAG	Mode of file

Contents	Meaning
X'00'	Input
X'01'	Output

Name	Description
FILENUM	File index in FILPTRS
IOCODE	Error Return code (X'01'
OPENFLG	

Contents	Meaning
X'01'	GET request
X'02'	PUT request
X'08'	OPEN request
X'29'	RESET request

EXITDISP
CURIN
BUFDIS
RECRDEND
RECCNT
ITEMCNT
PHYSBUF
FRSTBYTE
PHYSEND
LRECLMAX
BLOCKMAX
RESETFLG
EXECSWIT
INPRES

VFILTAB (File Table for Record Oriented Files)

000 (000) VNAMELN	002 (002) VFILNAME					
	014 (020) VF TYPE	VF MODE	016 (022) VFILE NO	VCEX CEPT	018 (024) VF ORG	VACS LAST	01A (026) VEXITDISP	01C (028) VERROR
020 (032) VACBA	024 (036) VAREA				028 (040) VAREALN			02C (044) VRKP
030 (048) VCKEYA	034 (052) VLKEYA				038 (056) VRPLA			03C (060) VAREALNR
040 (064) VAREALNW	044 (068) VAREALNP				048 (072) VARENDA			04C (076) VRBACUR
050 (080) VKEYLEN	052 (082) VRE QST	VKEY ED	054 (084 VCKEYLEN	056 (086) V COND	VL COND	058 (088) VLEX CEPT	VAAST REQ	05A (090) VLKEYED	05C (092) VLKEYLN
060 (096) VACBLEN	064 (100) VRPLLEN				068 (104) VEXLLEN			06C (108) VRBALAST
070 (112) VRBAEND	074 (116) VCLEARA				078 (120) VCLEARLN			07C (124) VAREAMNL
080 (128) VAREAMXL	084 (132) VLNAREA							

VFILTAB Table Entries

Name	Description
VNAMELN	Length of file name
VFILNAME	DD name of file
VFTYPE	Type of file

Contents	Meaning
X'00'	Stream file
X'01'	Record file (VSAM)
X'80'	Undefined file
X'40'	PRINT TO/INPUT FROM file

VFMODE	Mode of file

Contents	Meaning
X'01'	Input
X'02'	Output
X'03'	Input/output
X'04'	Hold
X'05'	OUT REUSE
X'06'	ALL REUSE
X'07'	ALL HOLD REUSE

VFILENO	File index in FILPTRS
VCEXCEPT	Error return code
VFORG	File organization

Contents	Meaning
X'00'	ESDS
X'01'	KSDS
X'02'	RRDS

VACSLAST	Last file access code
VEXITDSP	Displacement from EXTPTRS of exit entry
VERROR	Error code sent to error routine
VACBA	Address of ACB
VAREA	Address of record area
VAREALN	Length of record area
VRKP	Starting position of key in record
VCKEYA	Address of current key area
VLKEYA	Address of last key area
VRPLA	Address RPL
VAREALNR	Length of last record read
VAREALNW	Length of last record written

Name	Description
VAREALNP	Length of last record accessed
VARENDA	Address of last position in VAREA
VRBACUR	Relative byte address of record to be accessed
VKEYLEN	Length of record for file
VREQST	Current request
VKEYED	Current request keyed?

Contents	Meaning
0	No
1	Yes

VCKEYLN	Length of key in current request
VCOND	Key search condition in current request
VLCOND	Key search request in last request
VLEXCEPT	Last error return code
VLASTREQ	Last request
VLKEYED	Last request keyed?

Contents	Meaning
0	No
1	Yes

VLKEYLN	Length of key in last request
VACBLEN	Length of ACB
VRPLLEN	Length of RPL
VEXLLEN	Length of EXLIST
VRBALAST	Relative byte address of last record
VRBALND	File extent at open time
VCLEARA	First position of record to clear
VCLEARLN	Number of positions to clear
VAREANML	Minimum size of record
VAREAMXL	Maximum size of record
VLNAREA	

VARCON - VARIABLE AND CONSTANT AREA

The variable and constant area contains storage for the variables and constants that are defined in the current VS BASIC program. The first part of this area is pre-allocated for certain constants that the VS BASIC Processor requires.

VARCON - Variable and Constant Area (Compilation and Run-time)

00(000) INTCON or FLT0	08(008) FLT1
10(016) FLT2	18(024) FLT3
20(032) FLT4	28(040) FLT5
30(048) FLT6	38(056) FLT7
40(064) FLT8	48(072) FLT9
50(80) INTGRLN or E	58(088) PI
60(096) SQR2	68(104) INCM
70(112) LBKG	78(120) GAL1
80(128) FLT180PI	88(136) FLTPI180
90(144) FLTPLUS1	98(152) FLTMIN1
A0(160) FLT47	A8(168) FLT32767
B0(176) FLTFXM1	B8(184) FLT32
C0(192) FLT9S5	C8(200) FLT559
D0(208) FIX255 / D4(212)	E8(232) BUFF
F0(240) RELREC#	F8(248) ERRN
100(256) ERRL	108(264) ERRC
110(272) ERRF	
NULLSTR	
123(291)	
User Defined Variables and Constants	

Name	Description
INTCON or FLT0	Floating-point 0
FLT1	Floating-point 1
FLT2	Floating-point 2
FLT3	Floating-point 3
FLT4	Floating-point 4
FLT5	Floating-point 5
FLT6	Floating-point 6
FLT7	Floating-point 7
FLT8	Floating-point 8
FLT9	Floating-point 9
E	The constant e
PI	The constant pi
SQR2	The square root of 2
INCM	Conversion constant for inches to centimeters
LBKG	Conversion constant for pounds to kilograms
FLT180PI	Floating-point 180/pi
FLTPI180	Floating-point pi*180
FLTPLUS1	Floating-point +1
FLTMIN1	Floating-point -1
FLT47	Floating-point 47
FLT32767	Floating-point 32767 (32K)
FLTFXM1	Largest floating-point number minus 1
FLT32	Floating-point 32
FLT9S5	Conversion constant for centigrade to fahrenheit
FLT5S9	Conversion constant for fahrenheit to centigrade
FIX255	Fixed-point 255
NULLSTR	String of blanks
BUFF	Read-only variable &BUFF
RELREC#	Read-only variable &REC
ERRN	Read-only variable &ERR
ERRL	Read-only variable &LINE
ERRC	Read-only variable &CODE
ERRF	Read-only variable &FILE

OBJAREA - OBJECT AREA

The object area is the portion of storage that is set aside for the object code that is generated for the current VS BASIC program. The beginning of the area is pre-set to error branch addresses that may be required during execution. In most cases, register 10 points to OBJAREA. If the value in register 10 is lost, the address of the object area can be found in BSOBJ in PGR.

OBJAREA - Object Code Area (Compilation and Run-time)

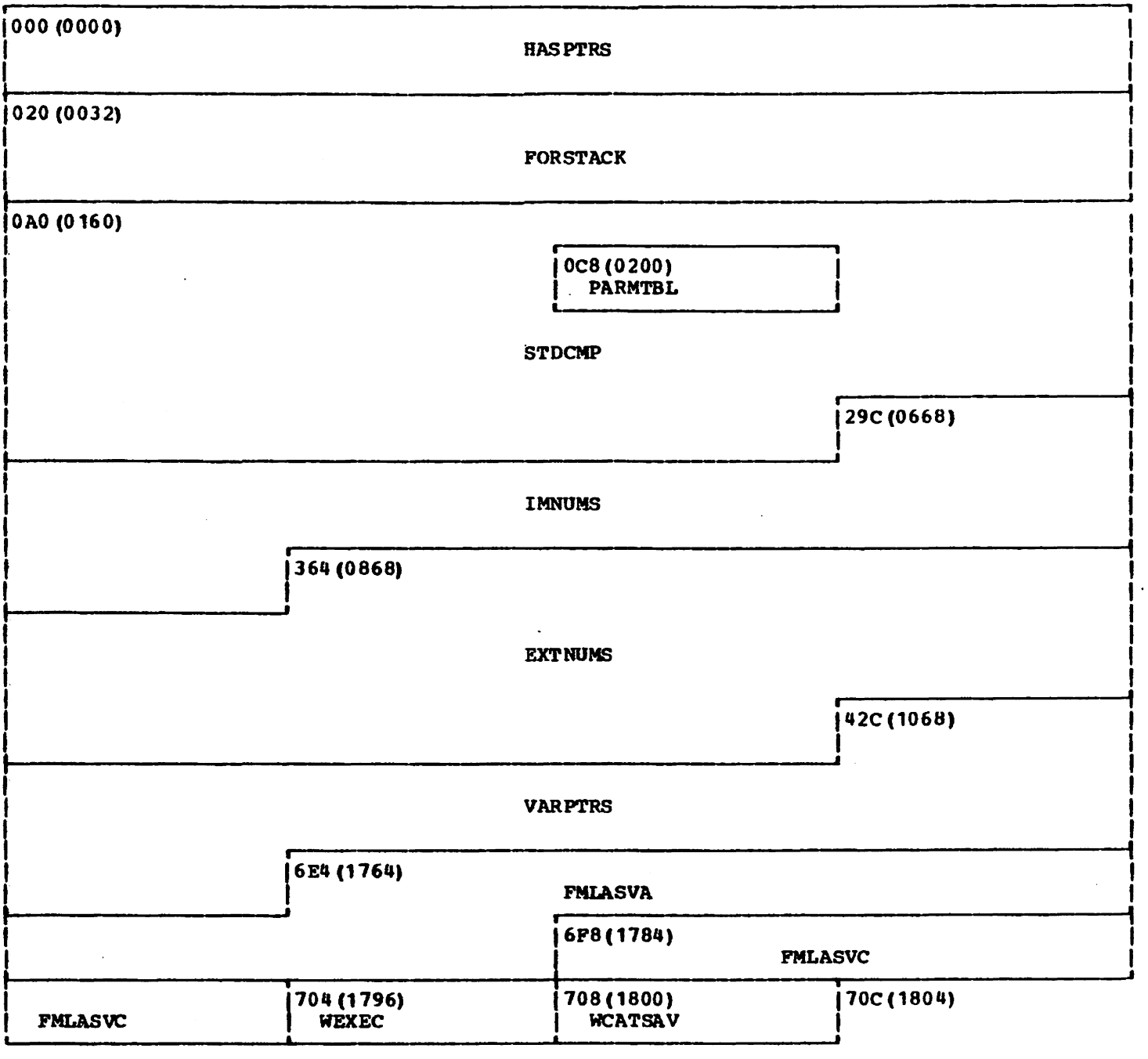
00(00) REFERR1	08(08) REFERR2
10(16) SUBSERR	18(24) GOSUBER or STKERR
20(32) RETURNER	28(40) SUBSCR2
30(48) SUBSCR3	38(56) CONCAT1
40(64) ARGERR	48(72) FENDERR
50(80)	5A(90)
72(114) BGEX	

Name	Description
REFERR1	Undefined line number branch address
REFERR2	Undefined user function branch address
SUBSERR	Subscript out of bounds branch address
GOSUBER or STKERR	DEF/GOSUB call to deeply nested branch address
RETURNER	Illegal DEF/GOSUB return branch address
SUBSCR2	Number of subscripts not equal branch address
SUBSCR3	Invalid subscript branch address
CONCAT1	Concatenated string too long branch address
ARGERR	Wrong argument type branch address
FENDERR	No return from function definition branch address
	Code to link to ICDKERR
	Code to ensure object code/library compatibility
BGEX	Address of start of object code

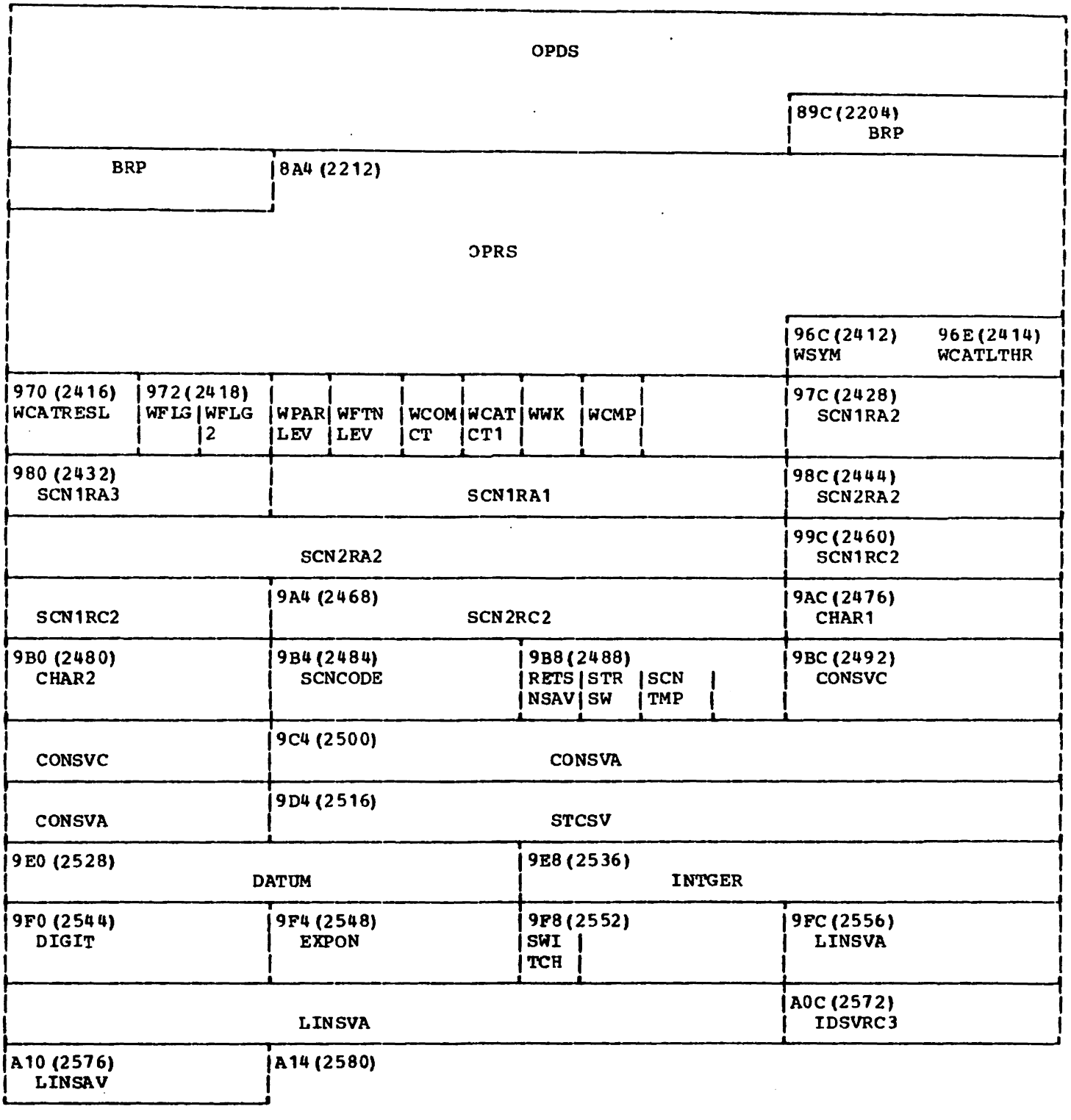
PRGA - COMPILER WORKSPACE

PRGA is a compile-time only area that contains dynamic workspace for compiler routines. The compiler routines must use this outside workspace because they are re-entrant. There is no base register for PRGA; each routine that uses it establishes its own base. The address of PRGA can, however, be found in BSPRGA in PRG.

PRGA - Work Areas (Compilation)



PRGA - Work Areas (Compilation) (continued)



Name	Description
HSHPTRS	HASH TABLE (FOR LINTAB)
FORSTACK	FOR STATEMENT STACK
STDCMP	STATEMENT PROCESSOR WORKAREA
PARMTBL	VARPTRS SAVE AREA USED DURING FUNCTION PROCESSING
	Contents: 0 — DISPLACEMENT INTO VARPTRS FOR PARAMETERS
	Contents: 2 — CONTENT OF VARPTRS ENTRY BEFORE PARAMETER ALLOCATION
IMNUMS	IMAGE ENTRIES
EXTNUMS	EXIT ENTRIES
VARPTRS	NUMERIC AND ALPHANUMERIC VARIABLE POINTERS
FMLASVA	SAVE AREA FOR ICDJFMLA ABSOLUTE REGISTERS
FMLASVC	COMPILER REGISTER SAVE AREA
WEXEC	INSTRUCTION EXECUTE WORD
WCATSAV	STRING CONCATENATION SAVE OPDS ENTRY
OPDS	OPERAND STACK
BRP	REGISTER TABLE
OPRS	OPERATOR STACK
WSYM	SAVE AREA FOR PREVIOUS OPERATOR OR OPERAND
WCATLTHR	STRING CONCATENATION RESULTANT LENGTH
WCATRESL	STRING CONCATENATION RESULTANT LOCATION
WFLG	ICDJFMLA ENTRY FLAG BYTE
	Contents: X'00' — RECEIVING VARIABLE ENTRY
	Contents: X'04' — SINGLE EXPRESSION ENTRY, RETURN CODE IN REG.
	Contents: X'03' — SINGLE EXPRESSION ENTRY, NUMERIC EXPRESSION RETURN IN FLOATING-POINT REGISTER 0
	Contents: X'01' — SINGLE EXPRESSION ENTRY, RETURN IN STORAGE
	Contents: X'0F' — SENDING LIST ENTRY
WFLG2	WORK FLAG BYTE
WPARLEV	PARENTHESIS LEVEL COUNTER
WFTNLEV	FUNCTION OR SUBSCRIPTED VARIABLE LEVEL COUNTER
WCOMCT	COMMA COUNTER
WCATCT1	STRING CONCATENATION - OPDS ENTRY TO PROCESSOR
WWK	WORK BYTE
WCMP	COMPARE OPERATOR SAVE AREA
SCN1RA2	ICDJSCN1 SAVE AREA FOR ABSOLUTE REGISTERS 2-5
SCN2RA2	ICDJSCN2 SAVE AREA FOR ABSOLUTE REGISTERS 2-5
SCN1RC2	ICDJSCN1 SAVE AREA FOR REGISTERS RC3-RC4
SCN2RC2	ICDJSCN2 SAVE AREA FOR REGISTERS RC3-RC4
CHAR1	TRANSLATED FIRST SOURCE CHARACTER
CHAR2	TRANSLATED SECOND SOURCE CHARACTER
SCNCODE	SAVE AREA FOR ARGUMENT LIST ENTRY
RETSWSAV	SAVE AREA FOR RETSW
STRSW	SUBSTRING PROCESSING SWITCH
	Contents: 0 — NO SUBSTRING ENCOUNTERED
	Contents: 1 — SUBSTRING BEING PROCESSED
SCNTMP	USED FOR FULL WORD ALIGNMENT
CONSVC	
CONSVA	
STCSV	
DATUM	
INTGER	
DIGIT	
EXPON	
SWITCH	
LINSVA	SAVE AREA FOR ABSOLUTE REGISTERS
IDSVRC3	SAVE AREA FOR REGISTER RC3
LINSAV	SAVE AREA FOR BINARY LINE NUMBER

NUC - COMMON COMPILER ROUTINES

NUC is the first block of the compiler. It is located in the compiler module ICDJNUCL. Register 7 always points to NUC. NUC contains routines that are commonly used by most of the other compiler routines. In addition, NUC contains a table of numeric constants and bit values, masks, a translate table for character conversion and a table of addresses for all of the compiler routines (ADDTBL).

NUC (First Block of the Compiler)

000 (000)	004 (004)	008 (008) $0	00C (012) $1
010 (016) $2	014 (020) $3	018 (024) $4	01C (028) $5
020 (032) $6	024 (036) $7	028 (040) $8	02C (044) $9
030 (048) $10	034 (052) $11	038 (056) $12	03C (060) $16
040 (064) $17	044 (068) $18	048 (072) $22	04C (076) $23
050 (080) $24	054 (084) $29	058 (088) $30	05C (092) $36
060 (096) $38	064 (100) $39	068 (104) $60	06C (108) $90
070 (112) $100	074 (116) $200	078 (120) $255	07C (124) $256
080 (128) $1000	084 (132) $2048	080 (136) $4095	08C (140) $4096
090 (144) $32767	094 (148) $64000	098 (152) B13	09C (156) B14
0A0 (160) B15	0A4 (164) B13A15	0A8 (168) B14T15	0AC (172) B20T31
0B0 (176) B16T31	0B4 (180) B0T28	0B8 (184) B0T29	0BC (188) B29T31
0C0 (192) SMASK1	0C4 (196) SMASK3	0C8 (200) LIFTMP	0CC (204) LFORTMP
0D0 (208) DKBFSZE	0D4 (212) VBASECHK	0D8 (216) CTAB FFFF FFFF	FF00 FFFF
FFFF FFFF	FFFF FFFF	FFFF FFFF	FFFF FFFF
FFFF FFFF	FFFF FFFF	FFFF FFFF	FFFF FFFF

NUC (First Block of the Compiler) (continued)

FFFF FFFF	FFFF FFFF	FFFF FFFF	FFFF FFFF
FFFF FFFF	FFFF FFFF	00FF FFFF	FFFF FFFF
FFFF FFFF	5A73 6A76	FFFF FFFF	FFFF FFFF
FFFF FF26	6974 FFFF	6B6C FFFF	FFFF FFFF
FFFF FFFF	FFFF 5BFF	FFFF FFFF	FFFF FFFF
FFFF FF25	27FF 5CFF	FFFF FFFF	FFFF FFFF
FFFF 75FF	5EFF FFFF	FFFF FFFF	FFFF FFFF
FFFF FFFF	FFFF FFFF	FFFF FFFF	FFFF FFFF
FFFF FFFF	FFFF 5FFF	FFFF FFFF	FFFF FFFF
FFFF FFFF	FFFF 5DFF	FF0B 0C0D	0E0F 1011
1213 FFFF	FFFF FFFF	FF14 1516	1718 191A
1B1C FFFF	FFFF FFFF	FFFF 1D1E	1F20 2122
2324 FFFF	FFFF FFFF	0102 0304	0506 0708

090A FFFF	FFFF FFFF	1D8 (472) LCCDEF	1DC (476) LCCMPA
1E0 (480) LCCTL	1E4 (484) LCDEF2	1E8 (488) LCDFRM	1EC (492) LCERRN
1F0 (496) LCERRP	1F4 (500) LCERRS	1F8 (504) LCERRT	1FC (508) LCFBN1
200 (512) LCFBN2	204 (516) LCFCAT	208 (520) LCFEXP	20C (524) LCFGEN
210 (528) LCFNE2	214 (532) LCFRM2	218 (536) LCFRM3	21C (540) LCFRM5
220 (544) LCFUNY	224 (548) LCINFO	228 (552) LCMATD	22C (556) LCNUC1
230 (560) LCDATA	234 (564) LCFOR	238 (568) LCFORM	23C (572) LCIMAG
240 (576) LCRDIM	244 (580) LCRETV	248 (584) LCRUNA	24C (588) LCUSTB
250 (592) LCREAD	254 (596) LCDIMG	258 (600) LCFNE1	25C (604) LCEXIT
260 (608) LCVRD	264 (612) LCIF1	268 (616) LCIF2	26C (620) LCDDAT
270 (624) LCCHAIN	274 (628) 04C3 C8C1	C9D5 4040	27C (636) LCCLOSE
280 (640) 04C3 D3D6	E2C5 4040	288 (648) LCDEF	28C (652) 02C4 C5C6
4040 4040	294 (660) LCDELETE	298 (664) 05C4 C5D3	C5E3 C540

NUC (First Block of the Compiler) (continued)

2A0(672) LCDIM	2A4(676) 02C4 C9D4	4040 4040	2AC(684) LCEND
2B0(688) 02C5 D5C4	4040 4040	2B8(696) LCDEXT	2BC(700) 03C5 E7C9
E340 4040	2C4(708) LCTFN	2C8(712) 01C6 D540	4040 4040
2D0(720) LCTFOR	2D4(724) 02C6 D6D9	4040 4040	2DC(732) LCGET
2E0(736) 02C7 C5E3	4040 4040	2E8(744) LCGOSUB	2EC(748) 04C7 D6E2
E4C2 4040	2F4(756) LCGOTO	2F8(760) 03C7 D6E3	D640 4040
300(768) LCIF	304(772) 01C9 C640	4040 4040	30C(780) LCINPUT
310(784) 04C9 D5D7	E4E3 4040	318(792) LCLET	31C(796) 02D3 C5E3
4040 4040	324(804) LCMAT	328(808) 02D4 C1E3	4040 4040
330(816) LCNEXT	334(820) 03D5 C5E7	E340 4040	33C(828) LCON
340(832) 01D6 D540	4040 4040	348(840) LCOPEN	34C(844) 03D6 D7C5
D440 4040	354(852) LCPAUSE	358(856) 04D7 C1E4	E2C5 4040
360(864) LCPRINT	364(868) 04D7 D9C9	D5E3 4040	36C(876) LCPUT
370(880) 02D7 E4E3	4040 4040	378(888) LCTRD	37C(892) 03D9 C5C1
C440 4040	384(900) LCREREAD	388(904) 05D9 C5D9	C5C1 C440
390(912) LCRESET	394(916) 04D9 C5E2	C5E3 4040	39C(924) LCRESTOR
3A0(928) 06D9 C5E2	E3D6 D9C5	3A8(936) LCRETURN	3AC(940) 05D9 C5E3
E4D9 D540	3B4(948) LCREWRIT	3B8(952) 06D9 C5E6	D9C9 E3C5
3C0(960) LCSTOP	3C4(964) 03E2 E3D6	D740 4040	3CC(972) LCUSE
3D0(976) 02E4 E2C5	4040 4040	3D8(984) LCWRITE	3DC(988) 04E6 D9C9
E3C5 4040	3E4(996) FFFF FFFF	FFFF FFFF	FFFF FFFF

3F0(1008)

NUC

Name	Description
$0	F'0'
$1	F'1'
$2	F'2'
$3	F'3'
$4	F'4'
$5	F'5'
$6	F'6'
$7	F'7'
$8	F'8'
$9	F'9'
$10	F'10'
$11	F'11'
$12	F'12'
$16	F'16'
$17	F'17'
$18	F'18'
$22	F'22'
$23	F'23'
$24	F'24'
$29	F'29'
$30	F'30'
$36	F'36'
$38	F'38'
$39	F'39'
$60	F'60'
$90	F'90'
$100	F'100'
$200	F'200'
$255	F'255'
$256	F'256'
$1000	F'1000'
$2048	F'2048'
$4095	F'4095'
$4096	F'4096'
$32767	F'32767'
$64000	F'64000'
B13	X'00040000'
B14	X'00020000'
B15	X'00010000'
B13A15	X'00050000'
B14T15	X'00030000'
B20T31	X'00000FFF'
B16T31	X'0000FFFF'
B0T28	X'FFFFFFF8'
B0T29	X'FFFFFFFC'
B29T31	X'00000007'
SMASK1	
SMASK3	
LIFTMP	
LFORTMP	
DKBFSZE	
VBASECHK	
CTAB	Scan translate table
LCCDEF	Displacement from ICDJNUCL to ICDJCDEF
LCCMPA	Displacement from ICDJNUCL to ICDJCMPA
LCCTL	Displacement from ICDJNUCL to ICDJCTL
LCDEF2	Displacement from ICDJNUCL to ICDJDEF2
LCDFRM	Displacement from ICDJNUCL to ICDJDFRM
LCERRN	Displacement from ICDJNUCL to ICDJERRN
LCERRP	Displacement from ICDJNUCL to ICDJERRP
LCERRS	Displacement from ICDJNUCL to ICDJERRS
LCERRT	Displacement from ICDJNUCL to ICDJERRT
LCFBN1	Displacement from ICDJNUCL to ICDJFBN1
LCFBN2	Displacement from ICDJNUCL to ICDJFBN2
LCFCAT	Displacement from ICDJNUCL to ICDJFCAT
LCFEXP	Displacement from ICDJNUCL to ICDJFEXP
LCFGEN	Displacement from ICDJNUCL to ICDJFGEN
LCFNE2	Displacement from ICDJNUCL to ICDJFNE2
LCFRM2	Displacement from ICDJNUCL to ICDJFRM2
LCFRM3	Displacement from ICDJNUCL to ICDJFRM3
LCFRM5	Displacement from ICDJNUCL to ICDJFRM5
LCFUNY	Displacement from ICDJNUCL to ICDJFUNY
LCINFO	Displacement from ICDJNUCL to ICDJINFO
LCMATD	Displacement from ICDJNUCL to ICDJMATD
LCNUC1	Displacement from ICDJNUCL to ICDJNUC1
LCDATA	Displacement from ICDJNUCL to ICDJDATA
LCFOR	Displacement from ICDJNUCL to ICDJFOR
LCFORM	Displacement from ICDJNUCL to ICDJFORM
LCIMAG	Displacement from ICDJNUCL to ICDJIMAG
LCRDIM	Displacement from ICDJNUCL to ICDJRDIM
LCRETV	Displacement from ICDJNUCL to ICDJRETV
LCRUNA	Displacement from ICDJNUCL to ICDJRUNA
LCUSTB	Displacement from ICDJNUCL to ICDJUSTB
LCREAD	Displacement from ICDJNUCL to ICDJREAD
LCDIMG	Displacement from ICDJNUCL to ICDJDIMG
LCFNE1	Displacement from ICDJNUCL to ICDJFNE1
LCEXIT	Displacement from ICDJNUCL to ICDJEXIT
LCVRD	Displacement from ICDJNUCL to ICDJVRD
LCIF1	Displacement from ICDJNUCL to ICDJIF1
LCIF2	Displacement from ICDJNUCL to ICDJIF2
LCDDAT	Displacement from ICDJNUCL to ICDJDATA
VBTAB or LCCHAIN	Displacement from ICDKNUCL to ICDJCHN

Name	Description
LCCLOSE	Displacement from ICDJNUCL to ICDJCLOS
LCDEF	Displacement from ICDJNUCL to ICDJDEFR
LCDELETE	Displacement from ICDJNUCL to ICDJVDEL
LCDIM	Displacement from ICDJNUCL to ICDJDIM
LCEND	Displacement from ICDJNUCL to ICDJEND
LCDEXT	Displacement from ICDJNUCL to ICDJDEXT
LCTFN	Displacement from ICDJNUCL to ICDJTFN
LCTFOR	Displacement from ICDJNUCL to ICDJTFOR
LCGET	Displacement from ICDJNUCL to ICDJGET
LCGOSUB	Displacement from ICDJNUCL to ICDJGOSB
LCGOTO	Displacement from ICDJNUCL to ICDJGOTO
LCIF	Displacement from ICDJNUCL to ICDJIF
LCINPUT	Displacement from ICDJNUCL to ICDJINPT
LCLET	Displacement from ICDJNUCL to ICDJLET
LCMAT	Displacement from ICDJNUCL to ICDJMATV
LCNEXT	Displacement from ICDJNUCL to ICDJNEXT
LCON	Displacement from ICDJNUCL to ICDJON
LCOPEN	Displacement from ICDJNUCL to ICDJOPEN
LCPAUSE	Displacement from ICDJNUCL to ICDJPAUS
LCPRINT	Displacement from ICDJNUCL to ICDJPRNT
LCPUT	Displacement from ICDJNUCL to ICDJPUT
LCTRD	Displacement from ICDJNUCL to ICDJTRD
LCREREAD	Displacement from ICDJNUCL to ICDJVRRD
LCRESET	Displacement from ICDJNUCL to ICDJRSET
LCRESTOR	Displacement from ICDJNUCL to ICDJRSTO
LCRETURN	Displacement from ICDJNUCL to ICDJRETN
LCREWRIT	Displacement from ICDJNUCL to ICDJVRWR
LCSTOP	Displacement from ICDJNUCL to ICDJSTOP
LCUSE	Displacement from ICDJNUCL to ICDJUSE
LCWRITEF	Displacement from ICDJNUCL to ICDJVWRT

BIFTAB - BRANCH INFORMATION TABLE

The Branch Information Table is a table of addresses of all the run-time routines. It provides the VS BASIC Processor and the object code with the means to get from routine-to-routine during execution. Register 13 usually points to BIFTAB. If the value in register 13 is lost, the location of BIFTAB can also be found in VSBLIB in PRG.

ICDKBFTB (Branch Information Table)

000 (000) LCKETAB	004 (004) LCKCHN	008 (008) LCKCLK	00C (012) LCKCLOS
010 (016) LCKCNVT	014 (020) LCKCPU	018 (024) LCKDAT	01C (028) LCKDET
020 (032) LCKDOT	024 (036) LCKERRR	028 (040) LCKERRS	02C (044) LCKERRT
030 (048) LCKETF2	034 (052) LCKETOF	038 (056) LCKFSCN	03C (060) LCKGET
040 (064) LCKIDX	044 (068) LCKINPT	048 (072) LCKINTP	04C (076) LCKJDY
050 (080) LCKKLN	054 (084) LCKKPS	058 (088) LCKLEN	05C (092) LCKMADD
060 (096) LCKMASN	064 (100) LCKMASR	068 (104) LCKMAT	06C (108) LCKMDSR
070 (112) LCKMIDN	074 (116) LCKMINV	078 (120) LCKMMUL	07C (124) LCKMSCA
080 (128) LCKMSUB	084 (132) LCKMTRN	088 (136) LCKNCPD	08C (140) LCKNUM
090 (144) LCKOPEN	094 (148) LCKOPN1	098 (152) LCKORGE	09C (156) LCKPLIN
0A0 (160) LCKPRD	0A4 (164) LCKPRNT	0A8 (168) LCKPUT	0AC (172) LCKRDM1
0B0 (176) LCKRDM2	0B4 (180) LCKREAD	0B8 (184) LCKRLN	0BC (188) LCKRND
0C0 (192) LCKRSET	0C4 (196) LCKRUNX	0C8 (200) LCKSTR	0CC (204) LCKSTRP
0D0 (208) LCKSUM	0D4 (212) LCKTIM	0D8 (216) LCKTOUT	0DC (220) LCKVCLS
0E0 (224) LCKVDEL	0E4 (228) LCKVOPN	0E8 (232) LCKVRD	0EC (236) LCKVRRD
0F0 (240) LCKVRST	0F4 (244) LCKVRWR	0F8 (248) LCKVWRT	0FC (252) LCKDBPR
100 (256) LCKVEND	104 (260) LCKMAX	108 (264) LCKMIN	10C (268) LCKSQR

ICDKBFTB (Branch Information Table) (continued)

110 (272) LCKSIN	114 (276) LCKCOS	118 (280) LCKSEC	11C (284) LCKCSC
120 (288) LCKTAN	124 (292) LCKCOT	128 (296) LCKHSN	12C (300) LCKHCS
130 (304) LCKHTN	134 (308) LCKASN	138 (312) LCKACS	13C (316) LCKATN
140 (320) LCKLOG	144 (324) LCKLGT	148 (328) LCKLTW	14C (332) LCKEXP
150 (336) LCKPWR	154 (340) LCKDMAX	158 (344) LCKDMIN	15C (348) LCKDSQR
160 (352) LCKDSIN	164 (356) LCKDCOS	168 (360) LCKDSEC	16C (364) LCKDCSC
170 (368) LCKDTAN	174 (372) LCKDCOT	178 (376) LCKDHSN	17C (380) LCKDHCS
180 (384) LCKDHTN	184 (388) LCKDASN	188 (392) LCKDACS	18C (396) LCKDATN
190 (400) LCKDLOG	194 (404) LCKDLGT	198 (408) LCKDLTW	19C (402) LCKDEXP
1A0 (416) LCKDPWR	1A4 (420) LCKRUNY	1A8 (424) LCKON	1AC (428) LCKCHR
1B0 (448) LCKTIO	1B4 (436) LCKVTIO	1B8 (440) LCKXX9	1BC (444) LCKXX8
1C0 (448) LCKXX7	1C4 (452) LCKXX6	1C8 (456) LCKXX5	1CC (460) LCKXX4
1D0 (464) LCKXX3	1D4 (468) LCKXX2	1D8 (472) LCKXX1	

ICDBIFTB

Name	Description
BIFTAB, LCSTART, or LCKETAB	Address of ICDKETAB
LCKCHN	Address of ICDKCHN
LCKCLK	Address of ICDKCLK
LCKCLOS	Address of ICDKCLOS
LCKCNVT	Address of ICDKCNVT
LCKCPU	Address of ICDKCPU
LCKDAT	Address of ICDKDAT
LCKDET	Address of ICDKDET
LCKDOT	Address of ICDKDOT
LCKERRR	Address of ICDKERRR
LCKERRS	Address of ICDKERRS
LCKERRT	Address of ICDKERRT
LCKETF2	Address of ICDKETF2
LCKETOF	Address of ICDKETOF
LCKFSCN	Address of ICDKFSCN
LCKGET	Address of ICDKGET
LCKIDX	Address of ICDKIDX
LCKMAT	Address of ICDKMAT
LCKMDSR	Address of ICDKMDSR
LCKMIDN	Address of ICDKMIDN
LCKMINV	Address of ICDKMINV
LCKMMUL	Address of ICDKMMUL
LCKMSCA	Address of ICDKMSCA
LCKMSUB	Address of ICDKMSUB
LCKMTRN	Address of ICDKMTRN
LCKNCPD	Address of ICDKNCPD
LCKNUM	Address of ICDKNUM
LCKOPEN	Address of ICDKOPEN
LCKOPN1	Address of ICDKOPN1
LCKORGE	Address of ICDKORGE
LCKPLIN	Address of ICDKPLIN
LCKPRD	Address of ICDKPRD
LCKPUT	Address of ICDKPUT
LCKRDM1	Address of ICDKRDM1
LCKRDM2	Address of ICDKRDM2
LCKREAD	Address of ICDKREAD
LCKRLN	Address of ICDKRLN
LCKRND	Address of ICDKRND
LCKRSET	Address of ICDKRSET
LCKRUNX	Address of ICDKRUNX

Name	Description
LCKINPT	Address of ICDKINPT
LCKINTP	Address of ICDKINTP
LCKJDY	Address of ICDKJDY
LCKKLN	Address of ICDKKLN
LCKKPS	Address of ICDKKPS
LCKLEN	Address of ICDKLEN
LCKMADD	Address of ICDKMADD
LCKMASN	Address of ICDKMASN
LCKMASR	Address of ICDKMASR
LCKVRRD	Address of ICDKVRRD
LCKVRWR	Address of ICDKVRWR
LCKVWRT	Address of ICDKVWRT
LCKDBPR	Address of ICDKDBPR
LCKVEND	Address of ICDKVEND
LCKMAX	Address of ICDKMAX
LCKMIN	Address of ICDKMIN
LCKSQR	Address of ICDKSQR
LCKSIN	Address of ICDKSIN
LCKCOS	Address of ICDKCOS
LCKSEC	Address of ICDKSEC
LCKCSC	Address of ICDKCSC
LCKTAN	Address of ICDKTAN
LCKCOT	Address of ACDKCOT
LCKHSN	Address of ICDKHSN
LCKHCS	Address of ICDKHCS
LCKHTN	Address of ICDKHTN
LCKASN	Address of ICDKASN
LCKACS	Address of ICDKACS
LCKATN	Address of ICDKATN
LCKLOG	Address of ICDKLOG
LCKLGT	Address of ICDKLGT
LCKLTW	Address of ICDKLTW
LCKEXP	Address of ICDKEXP

Name	Description
LCKSTR	Address of ICDKSTR
LCKSTRP	Address of ICDKSTRP
LCKSUM	Address of ICDKSUM
LCKTIM	Address of ICDKTIM
LCKTOUT	Address of ICDKTOUT
LCKVCLS	Address of ICDKVCLS
LCKVDEL	Address of ICDKVDEL
LCKVOPN	Address of ICDKVOPN
LCKVRD	Address of ICDKVRD
LCKPWR	Address of ICDKPWR
LCKDMAX	Address of ICDKDMAX
LCKDMIN	Address of ICDKDMIN
LCKDSQR	Address of ICDKDSQR
LCKDSIN	Address of ICDKDSIN
LCKDCOS	Address of ICDKDCOS
LCKDSEC	Address of ICDKDSEC
LCKDCSC	Address of ICDKDCSC
LCKDTAN	Address of ICDKDTAN
LCKDCOT	Address of ICDKDCOT
LCKDHSN	Address of ICDKDHSN
LCKDHCS	Address of ICDKDHCS
LCKDHTN	Address of ICDKDHTN
LCKDASN	Address of ICDKDASN
LCKDACS	Address of ICDKDACS
LCKDATN	Address of ICDKDATN
LCKDLOG	Address of ICDKDLOG
LCKDLGT	Address of ICDKDLGT
LCKDLTW	Address of ICDKDLTW
LCKDEXP	Address of ICDKDEXP
LCKDPWR	Address of ICDKDPWR
LCKDRUNY	Address of ICDKRUNY
LCKON	Address of ICDKON
LCKCHR	Address of ICDKCHR
LCKTIO	Address of ICDKTIO
LCKVTIO	A ddress of ICDKVTIO
LCKXX9	Spare slot
LCKXX8	Spare slot
LCKXX7	Spare slot
LCKXX6	Spare slot
LCKXX5	Spare slot
LCKXX4	Spare slot
LCKXX3	Spare slot
LCKXX2	Spare slot
LCKXX1	Spare slot

COMREGN - DEBUG COMMUNICATIONS REGION

The Debug Communications Region contains information for communicating within and between routines of the VS BASIC Debug Processor. This information includes addresses and flags sed by the debug routines.

COMREGN (Debug Communications Region)

000 (000) COMSTAB	004 (004) COMNVSTB	008 (008) COMCVSTB	00C (012) COMNASTB
010 (016) COMCASTB	014 (020) COMCURPU	018 (024) COMQUAL	01C (028) COMCURST
020 (032) COMLASST	024 (036) COMLSTPU	028 (040) COMDIRCN	02C (044) COMWHNCN
030 (048) COMSUBST	034 (052) COMATNEB	038 (056) COMENDEB	042 (060) COMATNLN
040 (064) COMKSCNP	044 (068) COMPUTLP	048 (072) COMPPTGT	04C (076) COMKPARP
050 (080) COMCPPL	054 (084) COMJUMP	058 (088) COM TRACE | COMFLAGS	05C (092) COMFLOW
060 (096) COMZFLGS	064 (100) COMLSTDR	068 (104) COMP8	06C (108) COMEFMTA
070 (112) COMIFMTA	074 (116) COMCNVTA		

COMREGN

Name	Description
COMSTAB	Address of the statement table
COMNVSTB	Address of the numeric variable symbol table
COMCVSTB	Address of the character variable symbol table
COMNASTB	Address of the numeric array symbol table
COMCASTB	Address of the character array symbol table
COMCURPU	Address of the directory of the current program unit
COMQUAL	Address of the directory of the current qualified program unit
COMCURST	Address of the statement table entry for the current statement
COMLASST	Address of the statement table entry for the last statement
COMLSTPU	Address of the directory of the previous program unit
COMDIRCN	Address of the directory chain
COMWHNCN	Address of the WHEN chain
COMSUBST	Address of the subroutine stack
COMATNEB	Attention event control block
COMENDEB	END event control block
COMATNLN	Address of the line that was entered after an attention interrupt

Name	Description
COMKSCNP	Address of the entry point in IKJSCAN
COMPUTLP	Address of the entry point in IKJPUTL
COMPPTGT	Address of the entry point in IKJPTGT
COMKPARP	Address of the entry point in IKJPARS
COMCPPL	Address of the command processor parameter list
COMJUMP	The target address of a GO TO statement or zero
COMTRACE	Trace information

Contents	Meaning
X'00'	Trace oft
X'04'	Trace user function
X'14'	Trace on

CONFLAGS Flags

Contents	Meaning
COMRUNFL	This bit is turned on when a RUN command is issued
COMNEXT	This bit is turned on when a NEXT command is issued
COMPURGE	This bit is turned on when debug output is permitted. Turned oft by ATTN

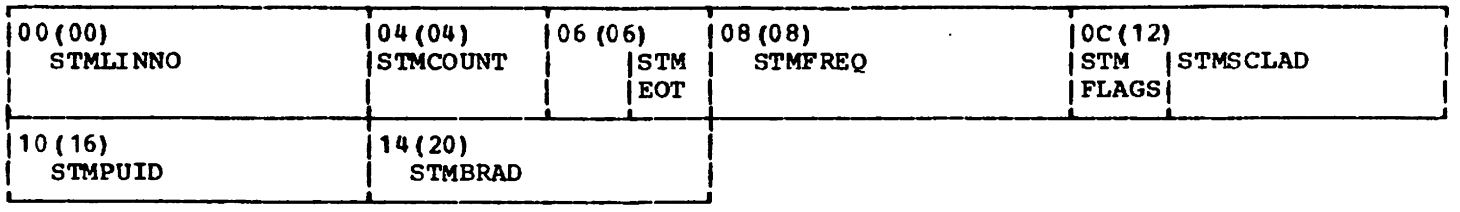
Contents	Meaning
COMSCLAB	This bit is turned on when a subcommand list is aborted because of an attention interrupt
COMNXTRG	This bit is turned on when a NEXT is issued if COMNINT is on
COMATNTG	This bit is turned on when an attention interrupt occurs and COMNINT is on
COMNINT	This bit is turned on by the monitor to indicate that attention interrupts are not acceptable
COMATSCL	This bit is turned on when ICDOBEY is called to execute a subcommand list
COMNOTIF	This bit is turned on when an AT notify message has been issued
COMINTXF	This bit is turned on if control is in an attention exit

Name	Description
	Contents: COMFINAL — Meaning: This bit is turned on when the current VS BASIC program has completed execution
	Contents: COMSCNOB — Meaning: This bit is turned on by ICDWNTST to signal that SCANOBEY must be called
	Contents: COMSTAI — Meaning: This bit is turned on in monitor STAI to signal an abnormal termination
	Contents: COMSL — Meaning: This bit is turned on if long precision is required
	Contents: COMCMS — Meaning: This bit is turned on if debug is operating under CMS
COMFLOW	Address of the flow table in the monitor
COMZFLGS	Address of the attention flag in Z#COMM
COMLSTDR	Previous directory entry
COMP8	Object code's register 8
COMEFMTA	Address of the E-conversion format
COMIFMTA	Address of the I-conversion format
COMCNVTA	Address of the convert switch

STMTABLE - DEBUG STATEMENT TABLE

The Debug Statement Table contains information about each source statement in the VS SIC program. It indicates whether an AT has been set for the statement and the appropriate AT parameters. STMTABLE is constructed by ICDBLDTB from information in LINTAB, LINPTRS, and LINCHN.

STMTABLE (Debug Statement Table)



STMTABLE

Name	Description
STMLNNO	Statement line number in binary
STMCOUNT	Counter for count keyword
STMEOT	End of table indicator if set to X'FF'
STMFREQ	Number of times this statement has executed
STMFLAGS	Flags

Contents	Meaning
STMATFLG	This bit is turned on if a breakpoint has been set at this statement
STMCNTFL	This bit is turned on if the count is other than one
STMNONEX	This bit is turned on if this is a non-executable statement
STMNOTFY	This bit is turned on if notify was requested
STMSCLFL	This bit is turned on if there is a subcommand list

STMSCLAD	Pointer to the subcommand list
STMPUID	Program unit identifier in EBCDIC
STMBRAD	Branch address

DIR - DEBUG UNIT DIRECTORY

The Debug Program Unit Directory contains the names of the program units in the current program and the names of the corresponding symbol or exception symbol table (see ICDNAME). This directory is built by ICDBLDTB from information in DEBUGPGM in PRG.

DIR (Debug Program Unit Directory)

0 (0)	4 (4)	8 (8)
DIRCHAIN	DIRNAME	DIREXSYM

DIR

Name	Description
DIRCHAIN	Pointer to the next entry or 0
DIRNAME	Name of the program unit
DIREXSYM	Pointer to the exception symbol table

ICDNAME - DEBUG SYMBOL TABLE

The Debug Symbol Table contains information about the scalar and array variables used in the current VS BASIC program. This table is built for a main program unit. An exception symbol table, with the same format, is built for each user function program unit in the program. The symbol tables are built by ICDBLDTB from information in VARCON and ARRYDSC.

ICDNAME (Debug Symbol Table Entry)

00 (00)		02 (02)	04 (04)	08 (08)		0A (10)		0C (12)
NAM LEN		NAMNAME	NAMADDR	NAM TYPE	NAM MODE	NAM NODIM	NAM EOT	NAMDIMS (1)
10 (16)								
NAMDIMS (2)								

ICDNAME

Name	Description
NAMLEN	Length of the character variable or array element
NAMNAME	Scalar or array name (left justified)
NAMADDR	Address of the variable or array in the object module
NAMTYPE	Type field, zero indicates a scalar and a non-zero indicates an array
NAMMODE	Mode of scalar or array element
NAMNODIM	Number of dimensions for an array
NAMEOT	End of table indicator
NAMDIMS1	Size of dimension
NAMDIMS2	Size of dimension

Table 9. Data Area Cross-reference Index (Part 1 of 6)

Symbol	Referred to in Module(s)
$AE	ICDJFUTS, ICDJNUC1
$AER	ICDJFUTS
$CE	ICDJFUTS, ICDJNUC5
$DE	ICDJFUTS, ICDJNUC1
$DER	ICDJFUTS
&LCER	ICDJFUTS
$LE	ICDJFUTS, ICDJNUC1, ICDJNUC2, ICDJRUNA
$LER	ICDJFUTS, ICDJNUC1
$LD	ICDJMATV
$LPER	ICDJFUTS
$LTER	ICDJFUTS, ICDJNUC2
$ME	ICDJFUTS, ICDJNUC1
$SE	ICDJFUTS, ICDJNUC1
$SER	ICDJFUTS
$STE	ICDJFUTS, ICDJNUC1, ICDJNUC2
$0	ICDJNUC1, ICDJMATV
$1	ICDJDEFR, ICDJMATV
$2	ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC5, ICDJMATV
$3	ICDJDEFR, ICDJNUCL, ICDJNUC1, ICDJVERB
$4	ICDJCMPA, ICDJDEFR, ICDJIOVB, ICDJNUCL, ICDJNUC2, ICDJNUC3, ICDJUSFN, ICDJMATV
$5	ICDJNUCL, ICDJNUC1
$6	ICDJNUC2, ICDJNUC3
$7	ICDJUSFN
$8	ICDJCMPA, ICDJDEFR, ICDJNUC1
$9	ICDJNUC5
$10	ICDJNUCL, ICDJNUC1
$11	ICDJNUC1, ICDJMATV
$12	ICDJCMPA
$16	ICDJDEFR, ICDJNUCL
$17	ICDJDEFR, ICDJNUCL, ICDJNUC1
$18	ICDJNUC1
$22	ICDJNUC1
$24	ICDJCMPA
$38	ICDJNUC1, ICDJMATV
$39	ICDJNUC1, ICDJMATV
$90	ICDJNUC5
$100	ICDJNUC1
$255	ICDJIOVB, ICDJNUC1
$256	ICDJCMPA, ICDJDEFR, ICDJFUTS, ICDJIOVB, ICDJNUC1
$1000	ICDJCMPA, ICDJDEFR
$2048	ICDJCMPA
$4095	ICDJUSFN
$4096	ICDJCMPA, ICDJDEFR, ICDJFUTS, ICDJNUCL, ICDJNUC1
$32767	ICDJNUC1, ICDJVERB
ABSWORK	ICDJNUCL
ARGADDR	ICDJIOVB, ICDJNUC1, ICDJNUC5, ICDKDSUB, ICDKFSCN, ICDKGET, ICDKGSUB, ICDKINPT, ICDKMAT, ICDKMINV, ICDKPRNT, ICDKREAD, ICDKSSUB, ICDKPUT
ARGCLTH	ICDKERR, ICDKGET, ICDKGSUB, ICDKINPT, ICDKPUT
ARGCODE	ICDJIOVB, ICDJNUC1, ICDJNUC5, ICDJVERB, ICDKDSUB, ICDKERR, ICDKFSCN, ICDKGET, ICDKGSUB, ICDKINPT, ICDKMAT, ICDKOPEN, ICDKPRNT, ICDKREAD, ICDKSSUB, ICDKPUT
ARGOPNCD	ICDKIOVB, ICDKCNVT, ICDKPLIN, ICDKPRNT
ARGPRCD	ICDJIOVB, ICDKCNVT, ICDKREAD
ARGRSCD	ICDJIOVB
ARRBYT	ICDJNUC1, ICDJRUNA
ARYDSCD1	ICDJNUC1, ICDJVERB, ICDKGET, ICDKINPT, ICDKMAT, ICDKMINV, ICDKREAD, ICDKPUT
ARYDSCD2	ICDJVERB, ICDKERR, ICDKGET, ICDKINPT, ICDKMAT, ICDKMINV, ICDKPRNT, ICDKREAD, ICDKPUT
ARYDSCL	ICDJNUC1, ICDKMAT, ICDKORGE, ICDJVERB, ICDKGET, ICDKINPT, ICDKPRNT, ICDKREAD, ICDKPUT

Table 9. Data Area Cross-reference Index (Part 2 of 6)

Symbol	Referred to in Module(s)
ARYDSCMX	ICDJNUC1, ICDKMAT, ICDKORGE
ARYDSCND	ICDJNUC1, ICDJVERB, ICDKMAT
ARYDSCOS	ICDJNUC1, ICDKGET, ICDKINPUT, ICDKMAT, ICDKMINV, ICDKPRNT, ICDKREAD, ICDKPUT
ALSADD	ICDJCMPA, ICDJFUTS, ICDJNUC1
ASLMASK	ICDJCMPA, ICDJFUTS, ICDJNUC1
ASVCCTL	ICDxEXEC
AVEXT	ICDJNUCL
AVFOR	ICDJCMPA, ICDJNUC2
AVIM	ICDJDEFR
AVLIN	ICDJNUCL
AVVAR	ICDJCMPA, ICDJNUCL, ICDJNUC1, ICDJRUNA
BASUSER	ICDxEXEC
BGEX	ICDJCMPA, ICDKERR
BIFTAB	ICDJERR, ICDJINFO, ICDJRUNA, ICDKERR, ICDKETOF, ICDKGSUB, ICDKINPT, ICDKINTP, ICDKMAT, ICDKMINV, ICDKORGE, ICDKREAD
BLKPAD	ICDKCNVT, ICDKPLIN, ICDKPRNT
BRP	ICDJFUTS, ICDJNUC1
BSDAT	ICDJDEFR, ICDJIOVB, ICDKREAD
BSEXTPTR	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDKERR
BSKEY	ICDJDEFR, ICDJIOVB
BLSINCHN	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJNUCL, ICDKERR
BSLINTAB	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJRUNA
BSLINTB	ICDJRUNA
BSLNPTRS	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJNUCL, ICDJNUC2, ICDJRUNA, ICDJUSFN, ICDKERR, ICDKFSCN, ICDKGET, ICDKGSUB, ICDKINPT, ICDKMAT, ICDKMINV, ICDKPRNT, ICDKREAD, ICDKPUT
BSOBJ	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJNUC5, ICDJRUNA, ICDJUSFN, ICDKCLOS, ICDKERR, ICDKFSCN, ICDKGET, ICDKGSUB, ICDKINPT, ICDKMAT, ICDKMINV, ICDKORGE, ICDJPRNT, ICDKREAD, ICDKPUT
BSOVFLW	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJRUNA, ICDKERR, ICDKMINV
BSPARMSV	ICDJCMPA, ICDJDEFR, ICDJUSFN
BSPRG	ICDJCMPA
BSPRGA	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJIOVB, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJNUC4, ICDJNUC5, ICDJRUNA, ICDJUSFN, ICDJVERB, ICDJMATV
BSSRC	ICDJCMPA, ICDJDEFR, ICDJNUCL
BSUFUN	ICDJCMPA, ICDJNUC1, ICDJUSFN, ICDKORGE
BSVARCN1	ICDJCMPA, ICDJERR, ICDJNCUL, ICDJNUC1, ICDJRUNA, ICDJVERB, ICDKERR, ICDKGSUB
BSVARCN2	ICDJCMPA, ICDJERR, ICDJNUCL, ICDJNUC1, ICDJRUNA, ICDJVERB, ICDKERR
BUFDIS	ICDxEXEC, ICDKCLOS, ICDKERR, ICDKETOF, ICDKGET, ICDKRSET, ICDKOPN1, ICDKPUT
BUFFPTR	ICDKINPT
BUFLTH	ICDxEXEC, ICDJCMPA, ICDJERR, ICDKERR, ICDKTOUT
BUFPTR	ICDxEXEC, ICDJERR, ICDKERR, ICDKTOUT
BOT28	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJNUC1, ICDJUSFN
BOT29	ICDJNUC1
B16T31	ICDJCMPA, ICDJERR
B20T31	ICDJNUC1, ICDJVERB
B29T31	ICDJCMPA
CALLER	ICDKETOF, ICDKGET, ICDKGSUB, ICDKINPT
CNOLINE	ICDJDEFR, ICDJNUCL
CNVTOUT	ICDKCNVT, ICDKPUT
CNVTRET	ICDKCNVT, ICDKPRNT, ICDKPUT
CONVENT	ICDJDEFR
CTLSV1	ICDJNUCL
CTLSV2	ICDJNUCL
CURBSREG	ICDJCMPA, ICDJNUCL, ICDJNUC1
CURCTL	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJNUC5, ICDJRUNA
CURDAT	ICDJDEFR, ICDJIOVB, ICDKREAD
CURDEF	ICDJDEFR
CURIN	ICDxEXEC, ICDKCLOS, ICDKERR, ICDKETOF, ICDKGET, ICDKRSET

Table 9. Data Area Cross-reference Index (Part 3 of 6)

Symbol	Refferred to in Module(s)
CURKEY	ICDJDEFR, ICDJIOVB, ICDKREAD
CURLINE	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJNUCL, ICDJNUC2
DATAKEYS	ICDJDEFR
DATUM	ICDJNUCL, ICDKETOF
DBUGNTRY	ICDJUSFN, ICDKORGE
DBUGTAB	ICDKORGE
DEFFORCT	ICDJNUC2, ICDJNUC3, ICDJUSFN
DEFNAME	ICDJNUCL, ICDJUSFN
DIGIT	ICDJNUCL, ICDKCNVT, ICDKETOF
DUPKYENT	ICDJDEFR
E	ICDKEXEC
EFILENAM	ICDKFSCN
EFMT	ICDKCNVT, ICDKPRNT, ICDKPUT
ENAMLNGTH	ICDKFSCN
ENDDAT	ICDJDEFR, ICDKREAD
ENDLIN	ICDJCMPA, ICDJNUCL, ICDJRUNA
ENDVAR	ICDJCMPA, ICDJNUCL, ICDJNUC1
EOFENT	ICDJDEFR
ERABS	ICDKERR
ERSVRET	ICDKERR
ERSVRET2	ICDKERR
EXITDISP	ICDKERR, ICDKGET, ICDKPUT
EXPON	ICDJNUCL, ICDKETOF
EXSW	ICDJDSUB, ICDKERR, ICDKGET, ICDKINPT, ICDKPRNT, ICDKSSUB, ICDKPUT
EXTEND	ICDJNUCL
EXTNUMS	ICDJDEFR, ICDJNUCL
FCNMRK	ICDKINTP
FFMT	ICDKRUNA, ICDKPRNT
FILENBR	ICDKEXEC, ICDKFSCN
FILEPTR	ICDJRUNA, ICDKCLOS, ICDKERR, ICDKFSCN
FIX255	ICDJFUTS
FLTFXM1	ICDJNUC1
FLTMIN1	ICDJFUTS
FLTPI180	ICDJFUTS
FLTPLUS1	ICDJFUTS
FLTO	ICDJFUTS, ICDJMATV
FLT1	ICDJMATV
FLT2	ICDKGSUB
FLT3	ICDKGSUB
FLT4	ICDKGSUB
FLT5	ICDKGSUB
FLT9	ICDKGSUB
FLT5S9	ICDJFUTS
FLT9S5	ICDJFUTS
FLT18CPI	ICDJFUTS
FLT32	ICDJFUTS
FLT47	ICDJFUTS
FLT32767	ICDJNUC1
FMLASVA	ICDJNUC1
FMTFLG	ICDKPLIN, ICDKPRNT
FORSTACK	ICDJNUC2, ICDJNUC3, ICDJRUNA
FORTMP	ICDJNUC2, ICDJNUC3
FPTMP	ICDJRUNA, ICDKERR, ICDKINTP, ICDKORGE
FSTDAT	ICDJDEFR, ICDJNUCL
FSTEXT	ICDJDEFR, ICDJNUCL
FSTFRM	ICDJDEFR, ICDJNUCL
GENFLAG	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJNUCL, ICDJNUC1, ICDJRUNA, ICDJUSFN
GOTOTMP	ICDJNUC4
GOTOTMP2	ICDJNUC4
H4	ICDJUSFN, ICDKCNVT
H8	ICDJUSFN, ICDKCNVT
IDSVRC3	ICDJNUCL
IFRTMPO	ICDJUNC5

Table 9. Data Area Cross-reference Index (Part 4 of 6)

Symbol	Referred to in Module(s)
IFRTMP1	ICDJNUC5
IFRTMP2	ICDJNUC5
IMEND	ICDJDEFR
IMNUMS	ICDJDEFR
INFOCD	ICDJNUC1
INFONAME	ICDJNUC1
INFONXT	ICDJNUC1
INFOTAB	ICDJNUC1
INPRES	ICDKETOF, ICDKGET
INGRLN	ICDNUC1
IOCNT	ICDJFUTS, ICDKGET, ICDKPRNT, ICDKPUT
IOCODE	ICDXEXEC, ICDKCLOS, ICDKERR, ICDKETOF, ICDKGET, ICDKRSET, ICDKPUT
IOERRENT	ICDJDEFR
ITEMCNT	ICDKGET, ICDKRSET, ICDKPUT
JDYTOD	ICDJRUNA, ICDKGSUB
KROUTRET	ICDJNUC1, ICDKGET, ICDKINPT, ICDKREAD, ICDJMATV
L#FLG3	ICDXEXEC, ICDJCMPA
L#NLINE	ICDXEXEC, ICDJCMPA
L#N2048	ICDXEXEC, ICDJCMPA, ICDJERR, ICDJRUNA, ICDKMINV
L#RTIME	ICDXEXEC, ICDKGSUB
L#SADDR	ICDXEXEC, ICDJCMPA
L#WIDTH	ICDXEXEC, ICDKORGE
LCCDEF	ICDJDEFR
LCCTL	ICDJCMPA, ICDJNUC5
LCDATA	ICDJDEFR
LCEND	ICDJNUCL
LCERRP	ICDJERR
LCEXIT	ICDJDEFR
LCFOR	ICDJNUCL
LCFORM	ICDJDEFR
LCIF1	ICDJNUC5
LCIF2	ICDJNUC5
LCIMAG	ICDJDEFR
LCINFO	ICDJNUC1
LCINPUT	ICDJMATV
LCKCLK	ICDKGSUB
LCKERRR	ICDKERR
LCKETOF	ICDKETOF
LCKINPT	ICDKINPT
LCKINTP	ICDKINTP
LCKMADD	ICDKMAT
LCMASR	ICDKMAT
LCKMAX	ICDJNUC1
LCKMIDN	ICDKMAT
LCKMIN	ICDKNUC1
LCKMINV	ICDKMINV
LCKRDM1	ICDKMAT
LCKREAD	ICDKREAD
LCKSTR	ICDKGSUB
LCKSUM	ICDKMAT
LCNUC1	ICDJCMPA
LCPAUSE	ICDJNUCL
LCPRINT	ICDJMATV
LCREM	ICDJNUCL
LCREREAD	ICDJMSTV
LCREWRIT	ICDJMATV
LCRESTOR	ICDJNUCL
LCRETURN	ICDJNUCL
LCRUNA	ICDJRUNA
LCSTOP	ICDJNUCL
LCUSTB	ICDJNUC1
LCWRITE	ICDNUCL

Table 9. Data Area Cross-reference Index (Part 5 of 6)

Symbol	Referred to in Module(s)
LETTMP	ICDJDEFR, ICDJNUC2
LEVM1ABS	ICDJGET, ICDKPUT
LEV0ABS	ICDJFUTS, ICDKCLOS, ICDKDSUB, ICDKGSUB, ICDKINTP, ICDKMAT, ICDKMINV, ICDKOPEN, ICDKPRNT, ICDKSSUB
LEV1ABS	ICDKCNVT, ICDKDSUB, ICDKETOF, ICDKFSCN, ICDKSSUB, ICDKOPN1
LEV3ABS	ICDKTOUT
LINCHN1	ICDJDEFR, ICDJNUCL
LINCHN2	ICDJDEFR
LINCHN3	ICDJDEFR, ICDJERR, ICDJNUCL, ICDKERR
LINFNO	ICDJDEFR, ICDJNUCL
LINSAV	ICDJNUCL
LINSRCE	ICDJDEFR
LINUM1	ICDJDEFR, ICDJERR, ICDJNUCL, ICDKERR
LINUM2	ICDJNUCL
LINWDTH	ICDKCNVT, ICDKORGE, ICDKPLIN, ICDKPRNT
LOCBRANC	ICDJUSFN
MASKM1	ICDKMAT, ICDKMINV
MASKM2	ICDKGSUB, ICDKMAT, ICDKMINV
MASK1	ICDJNUC4, ICDKGSUB, ICDKMAT, ICDKMINV, ICDKSSUB
MASK3	ICDJFUTS, ICDKCNVT, ICDKGSUB, ICDKMINV
MAXDGTS	ICDJRUNA, ICDKCNVT
MAXFILES	ICDXEXEC, ICDJRUNA
MAXNBLIN	ICDKPLIN, ICDKPRNT, ICDKTOUT
MIDCLK	ICDKGSUB
MODE	ICDJNUC5
MODEFLAG	ICDKCLOS, ICDKERR, ICDKGET, ICDKOPEN, ICDKRSET, ICDKPUT
MSGCNT	ICDKINTP
NBLIN	ICDKCNVT, ICDKPLIN, ICDKPRNT, ICDKTOUT
NOKEYENT	ICDJDEFR
NOLINE	ICDJCMPA, ICDJDEFR, ICDJNUCL
NULLSTR	ICDJNUC1, ICDKPRNT
OPDS	ICDJFUTS, ICDJNUC1
OPENFLG	ICDXEXEC, ICDKRSET
OPFLG	ICDJERR, ICDJRUNA, ICDKERR
OPRS	ICDJFUTS, ICDJNUC1
OPTIONS	ICDXEXEC, ICDJCMPA, ICDJERR, ICDJRUNA, ICDJUSFN, ICDKERR, ICDKORGE
PADCHAR	ICDJNUC2, ICDJNUC5, ICDJUSFN, ICDKERR, ICDKETOF, ICDKFSCN, ICDKMAT, ICDKREAD
PARMCNT	ICDJNUC1, ICDJUSFN
PARMPTR	ICDJCMPA, ICDJRUNA, ICDJUSFN
PARMTBL	ICDJNUC1, ICDJUSFN
PI	ICDJRUNA
PRG	ICDXEXEC, ICDJCMPA, ICDJDEFR, ICDJERR, ICDJFUTS, ICDJIOVB, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJNUC4, ICDJNUC5, ICDJRUNA, ICDJUSFN, ICDJVERB, ICDKATTN, ICDKCLOS, ICDKCNVT, ICDKDSUB, ICDKERR, ICDKETOF, ICDKFSCN, ICDKGET, ICDKGSUB, ICDKINPT, ICDKMAT, ICDKMINV, ICDKOPEN, ICDKORGE, ICDKPLIN, ICDKPRNT, ICDKREAD, ICDJRSET, ICDKSSUB, ICDKTOUT, ICDJMATV, ICDKOPN1, ICDKPUT
PRGA	ICDJCMPA, ICDJDEFR, ICDJERR, ICDJFUTS, ICDJIOVB, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJNUC4, ICDJNUC5, ICDJRUNA, ICDJUSFN, ICDJVERB, ICDKPUT
PSLTH	ICDXEXEC, ICDJNUC4, ICDJUSFN
PSMAX,	ICDJNUC4, ICDJUSFN
PSPTR	ICDXEXEC, ICDJCMPA, ICDJNUC4, ICDJUSFN
PSW1SV	ICDXEXEC
PSW2SV	ICDXEXEC, ICDJCMPA, ICDKINTP
RDECNO1	ICDKERR
RDECNO2	ICDKERR
RDIM1	ICDKMAT, ICDKMINV
RDIM2	ICDKMAT, ICDKMINV
RECCNT	ICDXEXEC, ICDKETOF, ICDKGET, ICDKRSET, ICDKPUT
RECRDEND	ICDXEXEC, ICDKETOF, ICDKGET, ICDKPUT

Table 9. Data Area Cross-reference Index (Part 6 of 6)

Symbol	Referred to in Module(s)
REDIM	ICDJMATV
RELWORK	ICDKFUTS, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDKGSUB
RESETFLG	ICDKGET, ICDKRSET, ICDKPUT
RETSW	ICDJDEFR, ICDJIOVB, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJUSFN, ICDJVERB, ICDJMATV
RETURNER	ICDJUSFN
RNDSEED	ICDKSSUB
SAVREG	ICDXEXEC
SCNCODE	ICDJNUC1
SCN1RP3	ICDJNUC1
SEQCHK	ICDJNUC1
SIGN	ICDKCNVT, ICDKPRNT
SLFORM	ICDJCMPA
SRCCR	ICDJNUCL
SRCLIN	ICDJIOVB, ICDJNUCL
SRCPTR	ICDJDEFR, ICDJFUTS, ICDJNUC1, ICDJNUCL
SSLFORM	ICDJCMPA, ICDJNUC1
STATESW	ICDJDEFR, ICDJERR, ICDJFUTS, ICDJIOVB, ICDJNUCL, ICDJNUC1, ICDJNUC2, ICDJNUC3, ICDJNUC4, ICDJNUC5, ICDJUSFN, ICDJVERB, ICDKERR, ICDJMATV
STCSV	ICDJDEFR, ICDJNUCL
STCTSV2	ICDKNUCL
STDCMP	ICDJDEFR, ICDJERR, ICDJIOVB, ICDJNUC2, ICDJNUC3, ICDJNUC4, ICDJNUC5, ICDJUSFN, ICDJVERB, ICDJMATV
STPRS	ICDJIOVB
STRSW	ICDJERR, ICDJNUC1
SUBSCR2	ICDJNUC1
SVCINST	ICDXEXEC
SWITCH	ICDXEXEC, ICDJNUCL, ICDKETOF
SWPFLG	ICDJRUNA
TMBUF	ICDXEXEC, ICDJERR, ICDKATTN, ICDKERR, ICDKINPT, ICDKTOUT
TOUTSW	ICDKERR, ICDKINPT, ICDKPRNT, ICDKTOUT
UFUNPTR	ICDJUSFN
UTTLOC	ICDXEXEC, ICDJCMPA, ICDJERR, ICDJRUNA, ICDKGSUB, ICDKMINV, ICDKORGE
VALRP2	ICDJNUC1
VALSW	ICDJNUC1
VAL4RA1	ICDJNUC1
VAROBASE	ICDJCMPA, ICDJNUCL, ICDJNUC1
VARCON	ICDJDEFR, ICDJFUTS, ICDJNUC1, ICDJRUNA, ICDKGSUB, ICDKPRNT, ICDJMATV
VARPTRS	ICDJNUC1, ICDJRUNA, ICDJUSFN
VSBATTN	ICDXEXEC, ICDJCMPA, ICDKORGE
VSBID	ICDXEXEC
VSBLIB	ICDXEXEC, ICDKATTN, ICDJRUNA, ICDKCNVT, ICDKDSUB, ICDKERR, ICDKETOF, ICDKGSUB, ICDKINTP, ICDKORGE, ICDKREAD, ICDKSSUB, ICDKOPN1, ICDKPUT
WBG	ICDJNUCL, ICDJUSFN
WCATCT1	ICDJFUTS
WCATLTHR	ICDJFUTS
WCATRESL	ICDJFUTS
WCATSAV	ICDJNUC1
WCMP	ICDJFUTS, ICDJNUC1
WCOMPT	ICDJFUTS, ICDJNUC1
WCUR	ICDJFUTS, ICDJNUCL, ICDJNUC1, ICDJUSFN
WEXEC	ICDJNUC1
WFLG	ICDJIOVB, ICDJNUC1, ICDJNUC2, ICDJNUC4, ICDJNUC5, ICDJUSFN, ICDJVERB, ICDJMATV
WFTNLEV	ICDJNUC1
WLTH	ICDJNUC1, ICDJRUNA, ICDJUSFN
WLTHSAV	ICDJUSFN
WSYM	ICDJNUC1
WWK	ICDJFUTS, ICDJNUC1

REGISTER CONVENTIONS

COMPILER REGISTER CONVENTIONS

Register	Symbolic	Usage
0	R0	Temporary absolute
1	RP4	Address of current source character
2	RA1	Absolute work register
3	RA2	Absolute work register
4	RA3	Absolute work register
6	RA4	Absolute work register
6	RA5	Absolute work register
7	RCBS	Address of ICDJNUCL
8	RC1	Address for compiler cover
9	RC2	Address for compiler cover
10	RC3	Address for compiler linkage
11	RCBS2	Address of ICDJNUC1
12	RPBS	Address of base of user area
13	RP1	Address for user area cover
14	RP2	Address for user area cover
15	RP3	Address of current object code

EXECUTION REGISTER CONVENTIONS

Register Number	Symbolic	Description
0	RRA0	Free absolute
1	RVAL1	Base address of VARCON1
2	RRFR	Free relocatable
3	RRA3	Free absolute
4	RRA4	Free absolute
5	RRA5	Free absolute
6	RVAL2	Base address of VARCON2
7	RARR	Base address of array storage
8	RRPBS	Base address of User Area
9	RLIN	Base address of LINPTRS table
10	ROBJ	Base address of Object Code
11	RRA11	Free absolute
12	RWKSP	Base address of Current Work Area
13	RRUN	Base address of Library
14	RLOC	Address of Current Object Code
15	RLINK	Address for Linkage Return

DIAGNOSTIC MESSAGES CROSS-REFERENCE INDEX

Table 10 lists the error message identifiers of the VS BASIC Processor. Messages produced are in the following ranges:

Executor	ICD000 - ICD199
Compiler	ICD200 - ICD299
Library	ICD400 - ICD499
Debug	ICD700 - ICD999
VSAM	ICD460 - ICD477 AND ICD481 - ICD484
Conversion Utility	ICD601 - ICD662
Renumbering Facility	ICD900-ICD912

The processor is identified with the following names:

VSPC Executor	ICDPxxxx
TSO Executor	ICDQxxxx
CMS Executor	ICDWxxxx
OS Executor	ICDYxxxx
DOS Executor	ICDZxxxx
Compiler	ICDJxxxx
Library	ICDKxxxx
Conversion Utility	ICDLUTIL
Renumbering Facility	ICDQRNMx
Debug	ICDxxxxx (other than those listed)

Table 10. Diagnostic Messages Directory (Part 1 of 7)

Message Number	Issued by: Modules or Entry Points	Component
ICD001	ICDQEXEC, ICDYEXEC, ICDZEXEC	Executor
ICD002	ICDWEXEC	Executor
ICD003	ICDWEXEC	Executor
ICD006	ICDWEXEC	Executor
ICD007	ICDWEXEC	Executor
ICD008	ICDQEXEC, ICDYEXEC, ICDZEXEC	Executor
ICD009	ICDQEXEC	Executor
ICD010	ICDQEXEC	Executor
ICD011	ICDQEXEC	Executor
ICD012	ICDQEXEC	Executor
ICD013	ICDQEXEC	Executor
ICD014	ICDZEXEC	
ICD015	ICDQEXEC, ICDYEXEC	Executor
ICD016	ICDQEXEC	Executor
ICD017	ICDQEXEC	Executor
ICD018	ICDZEXEC	
ICD019	ICDQEXEC	Executor
ICD020	ICDQEXEC	Executor
ICD022	ICDQEXEC	Executor
ICD023	ICDQEXEC	Executor
ICD024	ICDQEXEC	Executor
ICD025	ICDQEXEC	Executor
ICD026	ICDQEXEC	Executor
ICD027	ICDPEXEC, ICDQEXEC	Executor
ICD028	ICDQEXEC	Executor
ICD029	ICDQEXEC	Executor
ICD030	ICDQEXEC	Executor
ICD031	ICDQEXEC	Executor
ICD032	ICDQEXEC	Executor
ICD033	ICDQEXEC	Executor
ICD034	ICDPEXEC, ICDQEXEC, ICDYEXEC, ICDZEXEC	Executor

Table 10. Diagnostic Messages Directory (Part 2 of 7)

Message Number	Issued by	
	Modules or Entry Points	Component
ICD035	ICDQEXEC	Executor
ICD042	ICDQEXEC, ICDXEXEC	Executor
ICD043	ICDQEXEC	Executor
ICD044	ICDQEXEC	Executor
ICD045	ICDQEXEC	Executor
ICD046	ICDQEXEC	Executor
ICD047	ICDQEXEC	Executor
ICD051	ICDYEXEC	Executor
ICD052	ICDYEXEC, ICDZEXEC	Executor
ICD053	ICDYEXEC, ICDZEXEC	Executor
ICD054	ICDYEXEC, ICDZEXEC	Executor
ICD055	ICDYEXEC	Executor
ICD056	ICDYEYEC	
ICD058	ICDYEXEC, ICDZEXEC	Executor
ICD060	ICDYEXEC	Executor
ICD062	ICDYEXEC	Executor
ICD063	ICDYEXEC, ICDZEXEC	Executor
ICD065	ICDYEXEC, ICDZEXEC	Executor
ICD066	ICDYEXEC	Executor
ICD067	ICDYEXEC, ICDZEXEC	Executor
ICD068	ICDYEXEC, ICDZEXEC	
ICD069	ICDZEXEC	
ICD070	ICDZEXEC	
ICD081-ICD093	ICDPEXEC	Executor
ICD099	ICDQEXEC, ICDYEXEC, ICDPEXEC	Executor
ICD104	ICDWEXEC	Executor
ICD109	ICDWEXEC	Executor
ICD110	ICDWEXEC	Executor
ICD117	ICDWEXEC	Executor
ICD146	ICDWEXEC	Executor
ICD147	ICDWEXEC	Executor
ICD148	ICDWEXEC	Executor
ICD152	ICDWEXEC	Executor
ICD202	ICDJSCAN	Compiler
ICD203	ICDJCMPA, ICDJDEFR, ICDJNUCL, ICDJVAL	Compiler
ICD205	ICDJFMLA, ICDJFUTS	Compiler
ICD206	ICDJNUCL	Compiler
ICD208	ICDJNUCL	Compiler
ICD210	ICDJLET	Compiler
ICD211	ICDJIF	Compiler
ICD212	ICDJNEXT, ICDJRUNA, ICDJUSFN	Compiler
ICD213	ICDJNEXT	Compiler
ICD214	ICDJFOR	Compiler
ICD215	ICDJNEXT	Compiler
ICD216	ICDJFOR	Compiler
ICD217	ICDJNUCL	Compiler
ICD221	ICDJDEFR, ICDJFMLA, ICDJFOR, ICDJGOSB, ICDJGOTO, ICDJIF, ICDJIOVB, ICDJLET, ICDJMATV, ICDJNEXT, ICDJRDIM, ICDJUSFN, ICDJCHN, ICDJUSE	Compiler
ICD222	ICDJMATV	Compiler
ICD223	ICDJNUCL	Compiler
ICD224	ICDJIF, ICDJIOVB, ICDJMATV, ICDJUSFN, ICDJCHN, ICDJFUTS	Compiler
ICD225	ICDJFMLA	Compiler
ICD226	ICDJFMLA	Compiler
ICD227	ICDJFUTS	Compiler
ICD228	ICDJFMLA, ICDJUSE	Compiler
ICD229	ICDJNUCL	Compiler

Table 10. Diagnostic Messages Directory (Part 3 of 7)

Message Number	Issued by	
	Modules or Entry Points	Component
ICD230	ICDJFMLA, ICDJMATV, ICDJFUTS	Compiler
ICD231	ICDJFMLA	Compiler
ICD232	ICDJFMLA	Compiler
ICD233	ICDJFMLA	Compiler
ICD234	ICDJMATV	Compiler
ICD235	ICDJAADJ	Compiler
ICD236	ICDJFMLA	Compiler
ICD237	ICDJFMLA, ICDJDIM	Compiler
ICD238	ICDJMATV	Compiler
ICD239	ICDJMATV	Compiler
ICD240	ICDJFMLA	Compiler
ICD241	ICDJFMLA	Compiler
ICD242	ICDJFMLA	Compiler
ICD243	ICDJFMLA	Compiler
ICD244	ICDJUSFN, ICDJFMLA, ICDJFUTS, ICDJCHN	Compiler
ICD245	ICDJFMLA	Compiler
ICD246	ICDJFMLA	Compiler
ICD248	ICDJMATV	Compiler
ICD250	ICDJGOSB, ICDJGOTO, ICDJIOVB, ICDJNUCL	Compiler
ICD251	ICDJNEXT, ICDJFOR, ICDJNUCL, ICDJUSFN, ICDJDIM, ICDJUSE	Compiler
ICD256	ICDJNUCL, ICDJDIM, ICDJUSE	Compiler
ICD257	ICDJUSFN	Compiler
ICD258	ICDJUSFN	Compiler
ICD259	ICDJUSFN	Compiler
ICD260	ICDJUSFN	Compiler
ICD261	ICDJUSFN	Compiler
ICD262	ICDJUSFN	Compiler
ICD263	ICDJMATV	Compiler
ICD264	ICDJNUCL	Compiler
ICD265	ICDJIF	Compiler
ICD266	ICDJIF	Compiler
ICD267	ICDJDEFR	Compiler
ICD268	ICDJNUCL	Compiler
ICD269	ICDJDEFR, ICDJNUCL	Compiler
ICD270	ICDJDEFR	Compiler
ICD271	ICDJDEFR	Compiler
ICD272	ICDJDEFR	Compiler
ICD273	ICDJUSFN	Compiler
ICD274	ICDJDEFR	Compiler
ICD275	ICDJDEFR	Compiler
ICD277	ICDJIOVB	Compiler
ICD278	ICDJLET	Compiler
ICD401	ICDKON	Library
ICD402	Object Code	
ICD404	Object Code	
ICD405	Object Code	
ICD409	Object Code	
ICD410	Object Code	
ICD411	ICDPEXEC, ICDQEXEC, ICDWEXEC	Executor
ICD412	ICDKINTP	Library
ICD413	ICDKINTP	Library
ICD414	ICDKINTP	Library
ICD415	ICDKINTP	Library
ICD416	ICDKERR	Library
ICD417	ICDKERR	Library
ICD418	ICDKREAD	Library
ICD419	ICDKREAD	Library
ICD420	ICDKOPN1, ICDPPENT	Library, Executor
ICD422	ICDKOPN1, ICDPPENT	Library, Executor
ICD424	ICDKCLOS, ICKDETOF, ICDKGET, ICDKOPN1, ICDKPUT, ICDKRSET	Library
	ICDPVENT, ICDPPENT, ICDPCLS, ICDQVERR, ICDZVERR	Executor

Table 10. Diagnostic Messages Directory (Part 4 of 7)

Message Number	Issued by: Modules or Entry Points	Component
ICD425	ICDKCLOS, ICSKRSET, ICDPPENT	Library
ICD426	ICDKOPEN, ICDKPUT, ICDKRSET, ICDKTIO	Library, Executor
ICD427	ICDKETOF, ICDKGET, ICDPVENT, ICDPPENT, ICDQVERR, ICDZVERR	Library, Executor
ICD428	ICDKGET, ICDKPUT	Library
ICD429	ICDKGET, ICDKTIO	Library
ICD430	ICDKPUT, ICDKTIO	Library
ICD431	ICDKGET	Library
ICD432	ICDKGET	Library
ICD433	ICDKPRNT	Library
ICD434	ICDKPRNT	Library
ICD435	ICDKPRNT	Library
ICD436	ICDKPRNT	Library
ICD437	ICDKPRNT	Library
ICD438	ICDKPRNT	Library
ICD439	ICDKPRNT	Library
ICD440	ICDKINPT	Library
ICD441	ICDKMAT, ICDKDAT, ICDKIDX, ICDKJDY, ICDKLEN, ICDKNUM, ICDKRND, ICDKSTR	Library
ICD442	ICDKRND, ICDKLEN, ICDKJDY, ICDKNUM, ICDKDAT, ICDKDMAX, ICDKDMIN, ICDKSTR, ICDKIDX	Library
ICD443	ICDKDCSC, ICDKDASN, ICDKDACS, ICDKDTAN, ICDKDCOT, ICDKDSEC, ICDKDEXP, ICDKDSIN, ICDKDCOS, ICDKDHSN, ICDKDNCS	Library
ICD444	ICDKDSQR	Library
ICD445	ICDKDLOG, ICDKDLTW, ICDKLGT	Library
ICD446	ICDKDTAN, ICDKDCOT, ICDKDCSC, ICDKDSEC	Library
ICD447	ICDKDPWR	Library
ICD450	ICDKMAT	Library
ICD451	ICDKMAT	Library
ICD452	ICDKMAT, ICDKMINV	Library
ICD454	ICDKMAT, ICDKMINV	Library
ICD455	ICDKMINV	Library
ICD460	ICDKVIOR, ICDKTIO	Library
ICD461	ICDKVIOR	Library
ICD462	ICDQVOPN, ICDZVOPN, ICDPOPN	Executor
ICD463	ICDKVIOR	Library
ICD464	ICDKVIOR	Library
ICD465	ICDKVIOR	Library
ICD466	ICDKVIOR, ICDKTIO	Library
ICD467	ICDKVIOR	Library
ICD468	ICDKVIOR	Library
ICD469	ICDKVIOR, ICDKTOUT, ICDKINPT	Library
ICD470	ICDKVIOR	Library
ICD471	ICDKVIOR	Library
ICD472	ICDKTIO	Library
ICD473	ICDKVIOR	Library
ICD474	ICDKVIOR	Library
ICD475	ICDKVIOR	Library
ICD476	ICDKVIOR	Library
ICD477	ICDKVIOR	Library
ICD478	ICDKRLN, ICDKKPS, ICDKKLN	Library
ICD479	ICDKKPS, ICDKKLN, ICDKRLN	Library
ICD480	ICDKKLN, ICDKKPS	Library
ICD481	ICDQVCLS, ICDZVCLS, ICDPCLS	Executor
ICD482	ICDQVERR, ICDPVENT, ICDZVERR	Executor
ICD483	ICDQVERR, ICDPVENT, ICDZVERR	Executor
ICD484	ICDQVERR, ICDPVENT, ICDZVERR	Executor
ICD485	ICDKCLOS, ICDKGET, ICDKPUT, ICDKRSET	Library
ICD486	ICDKOPN1	Library

Table 10. Diagnostic Messages Directory (Part 5 of 7)

Message Number	Issued by	
	Modules or Entry Points	Component
ICD488	ICDKFSCN	Library
ICD490	ICDKMAT	Library
ICD491	ICDKORGE	Library
ICD492	ICDPPENT	Executor
ICD493	ICDPPENT	Executor
ICD494	ICDKIOVB, ICDPOPN	Library, Executor
ICD495	ICDKIOVB, ICDPOPN	Library, Executor
ICD496	ICDKIOVB, ICDPOPN	Library, Executor
ICD497	ICDKIOVB, ICDPOPN	Library, Executor
ICD498	ICDKIOVB, ICDPOPN	Library, Executor
ICD499	ICDKIOVB, ICDPOPN, ICDPPENT	Library, Executor
ICD500	ICDKIOVB, ICDPOPN	Library, Executor
ICD501	ICDKIOVB, ICDPOPN	Library, Executor
ICD502	ICDPPENT, ICDKVIOR	Library, Executor
ICD503	ICDKMINV	Library
ICD504	ICDKIOVB	Library
ICD505	ICDPPENT	Executor
ICD506	ICDPPENT	Executor
ICD507	ICDPOPN	Executor
ICD508	ICDPOPN	Executor
ICD509	ICDPOPN, ICDPCLS, ICDPVENT	Executor
ICD510	ICDKIOVB, ICDPPENT	Library, Executor
ICD511	ICDPPENT	Executor
ICD512	ICDPOPN	Executor
ICD513	ICDKUNDF	Library
ICD514	ICDKUNDF	Library
ICD515	ICDKINPT	Library
ICD516	ICDKINPT	Library
ICD517	ICDPPENT, ICDKVIOR	Executor, Library
ICD518	ICDPPENT, ICDKVIOR	Executor, Library
ICD519	ICDPPENT, ICDKVIOR	Executor, Library
ICD520	ICDPPENT, ICDKVIOR	Executor, Library
ICD601	ICDLUTIL	Utility
ICD602	ICDLUTIL	Utility
ICD607	ICDLUTIL	Utility
ICD610	ICDLUTIL	Utility
ICD662	ICDLUTIL	Utility
ICD700	ICDMSSG	Debug
ICD701	ICDMSSG	Debug
ICD702	ICDLISTO	Debug
ICD707	ICDLISTO	Debug
ICD709	ICDCHAIN	Debug
ICD710	ICDADRES	Debug
ICD711	ICDADRES	Debug
ICD721	ICDONITR	Debug
ICD722	ICDONITR	Debug
ICD723	ICDONITR	Debug
ICD725	ICDONITR	Debug

Table 10. Diagnostic Messages Directory (Part 6 of 7)

Message Number	Issued by	
	Modules or Entry Points	Component
ICD726	ICDONITR	Debug
ICD727	ICDONITR	Debug
ICD728	ICDSCAN	Debug
ICD730	ICDATTN, ICDSCAN	Debug
ICD731	ICDATTN, ICDSCAN	Debug
ICD732	ICDLSSCN	Debug
ICD733	ICDLSSCN	Debug
ICD734	ICDLSSCN	Debug
ICD735	ICDIDCHK	Debug
ICD736	ICDIDCHK	Debug
ICD737	ICDPGMCK	Debug
ICD738	ICDWNSCN	Debug
ICD739	ICDCDSCN	Debug
ICD740	ICDISCAN	Debug
ICD741	ICDISCAN	Debug
ICD742	ICDTSCN	Debug
ICD743	ICDTSCN	Debug
ICD744	ICDADRES	Debug
ICD745	ICDVSCN	Debug
ICD746	ICDVSCN	Debug
ICD747	ICDTSCN	Debug
ICD748	ICDTSCN	Debug
ICD749	ICDSSCN	Debug
ICD750	ICDSSCN	Debug
ICD751	ICDSSCN	Debug
ICD752	ICDSTSCN	Debug
ICD753	ICDSTSCN	Debug
ICD754	ICDSTSCN	Debug
ICD755	ICDLSSCN	Debug
ICD756	ICDLSSCN	Debug
ICD757	ICDLSSCN	Debug
ICD758	ICDVSCN	Debug
ICD759	ICDCDSCN	Debug
ICD760	ICDCDSCN	Debug
ICD761	ICDCDSCN	Debug
ICD762	ICDCDSCN	Debug
ICD763	ICDCDSCN	Debug
ICD764	ICDWNSCN	Debug
ICD765	ICDWNSCN	Debug
ICD766	ICDWNSCN	Debug
ICD768	ICDWNSCN	Debug
ICD769	ICDWNSCN	Debug
ICD770	ICDWNSCN	Debug
ICD771	ICDSTSCN	Debug
ICD772	ICDSTSCN	Debug
ICD773	ICDTSCN	Debug
ICD775	ICDSSCAN	Debug
ICD780	ICDLSSCN, ICDPRSCN	Debug
ICD795	ICDSTSCN	Debug
ICD800	ICDGOGO	Debug
ICD802	ICDGOGO	Debug
ICD802	ICDGOGO	Debug
ICD803	ICDONITR	Debug
ICD870	ICDWNTST	Debug
ICD871	ICDWNTST	Debug
ICD875	ICDWHENO	Debug
ICD876	ICDWHENO	Debug
ICD880	ICDOFFWO	Debug
ICD881	ICDOFFWO	Debug
ICD900	ICDQRNMS	RENUM

Table 10. Diagnostic Messages Directory (Part 7 of 7)

Message Number	Issued by	
	Modules or Entry Points	Component
ICD901	ICDQRNMS	RENUM
ICD902	ICDQRNMS	RENUM
ICD903	ICDQRNMS	RENUM
ICD904	ICDQRNMS	RENUM
ICD905	ICDQRNMS	RENUM
ICD906	ICDQRNMS	RENUM
ICD907	ICDQRNMS	RENUM
ICD908	ICDQRNMS	RENUM
ICD909	ICDQRNMS	RENUM
ICD910	ICDQRNMS	RENUM
ICD911	ICDQRNMS	RENUM
ICD912	ICDQRNMS	RENUM
ICD920	ICDSETO	Debug
ICD940	ICDLFQO	Debug
ICD941	ICDLFQO	Debug
ICD942	ICDLFQO	Debug
ICD943	ICDLBKO, ICDLFQO, ICDLISTO	Debug
ICD945	ICDLFQO	Debug
ICD947	ICDLFQO	Debug
ICD950	ICDLBKO	Debug
ICD952	ICDLBKO	Debug
ICD953	ICDLBKO	Debug
ICD955	ICDLBKO	Debug
ICD957	ICDLBKO	Debug
ICD990	ICDWHRO	Debug
ICD991	ICDTBACK	Debug
ICD992	ICDTBACK	Debug
ICD995	ICDWHRO	Debug
ICD996	ICDFLOW	Debug
ICD997	ICDFLOW	Debug

Table 11. VS BASIC Processor DOS/VS System Abnormal Termination Codes (Part 1 of 2)

Hexadecimal Representation	Specific Abnormal Termination Code Meaning
0C	Run time program check
10	Normal EOJ
11	No channel program translation for unsupported device
12	Insufficient buffer space for channel program translation
13	CCW with count greater than 32K
14	Page pool too small
15	Page fault in disabled program (not a supervisor routine)
16	Page fault in MICR stacker or page fault appendage routine
17	Main task issued a CANCEL macro with subtask still attached
18	Main task issued a DUMP macro with subtask still attached
19	Operator replied cancel as the result of an I/O error message
1A	An I/O error has occurred (see interrupt status information)
1B	Channel failure
1C	CANCEL ALL macro issued in another task
1D	Main task terminated with subtask still attached
1E	A DEQ macro was issued for a resource but tasks previously requesting a resource cannot be found because their save areas (containing register 0) were modified
1F	CPU failure
20	A program check occurred
21	An invalid SVC was issued by the problem program or macro
22	Phase not found in the core image library
23	CANCEL macro issued
24	Canceled due to an operator request
25	Invalid virtual storage address given (outside partition)
26	SYSxxx not assigned (unassigned LUB code)
27	Undefined logical unit
28	QTAM cancel in progress

Table 11. VS BASIC Processor DOS/VS System Abnormal Termination Codes (Part 2 of 2)

Hexadecimal Representation	Specific Abnormal Termination Code Meaning
29	Relocatable phase fetched or loaded by a supervisor without relocating loader support
2A	I/O error on page data set
2B	I/O error during fetch from private core image library
2C	Page fault appendage routine passed illegal parameter to supervisor
2D	Program cannot be executed/restarted due to a failing storage block
2E	Invalid resource request (possible deadlock)
2F	More than 255 PFIX requests for one page
30	Read past a /& statement
31	I/O error queue overflow during system error recovery procedure
32	Invalid DASD address
33	No long seek on a DASD
35	Job control open failure
36	Page fault in I/O appendage routine
38	Wrong privately translated CCW
39	Reserved
9C	Invalid parameter list item for I/O runtime (in module ICDKIOVB)
9F	Invalid parameter list item for run-time READ (in module ICDKREAD)
FF	Unrecognized cancel code

Table 12. VS BASIC Processor OS/VS System Abnormal Termination Codes

Decimal Representation	Specific Abnormal Termination Code Meaning
012	Run-time program check
156	Invalid parameter list item for run-time I-O (in module (ICDKIOVB)
158	Invalid parameter list item for run-time READ (in module ICDKREAD)
333	Object code and library incompatibile

DIAGNOSTIC PROCEDURES

In most cases, the processor produced diagnostic messages will direct you to the cause of the error. The message number indicates the component and module that detected the error. Refer to Table 10 in this section of the book for a list of the message numbers and the component and module that detected the error.

In some cases, particularly system errors and failures in the object code, the cause of the error is not readily apparent. In these cases, you should, at the very least, rerun the program to obtain a dump of storage, if one was not already produced. The dump is a useful source of information in your effort to isolate failures. If you have an interactive system available, (for example, TSO, VSPC, or CMS,) you can use their built-in diagnostic facilities to further aid your search.

OBTAINING A DUMP

Under OS/VS Including TSO

You may obtain a printed listing of a storage dump by rerunning the program and including a SYSUDUMP or SYSABEND DD statement in the JCL entered through the card reader. Under TSO, your logon procedure must allocate this data set or you may enter it dynamically prior to execution of the program.

Under CMS

You may obtain a printed listing of a storage dump by rerunning the program and then issuing a DEBUG command followed by a DUMP command.

Under VSPC

When severe VSPC and/or VS BASIC errors occur, VSPC writes messages ASU669 and ASU670 to the VSPC offline and online log as URGENT messages. If the SYSDUMP data set is available, the workspace is dumped and the dump identifier is printed in message ASU670. The format of the messages is as follows:

```
ASU669I time usernum PROCESSOR ERROR
        {BASIC|LBASIC} wsname

ASU670 ICDxxx (VS BASIC message text)

ASU670 {ICD085 DUMP ID:083}
       {ICD086 NO DUMP}
```

Under DOS/VS

You may obtain a printed listing of a storage dump by rerunning the program and specifying the DUMP option on the OPTION control card.

ERROR DIAGNOSIS USING A DUMP

1 Locate registers 8 and 12.

2 Add the hexadecimal value 240 to the contents of each register. The resulting value in one of these registers will point to the beginning of the Communication Region (PRG) indicated by the character string:

   VSBRECOG

   If register 8 points to the beginning of this character string, the program was in execution; if register 12 points there, the program was in compilation. (For CMS, if register 10 points there, the program was in the Executor code).

3 Locate the statement at which the program failed. If it failed during execution go step 7 and continue.

4 Locate register 1 (RP1) and 15 (RP3). During compilation register 1 usually points to the statement that is currently being processed and register 15 points to the corresponding object code that is generated. If these registers locate the source statement and its object code correctly, go to step 6 and continue. If they do not locate the correct statements continue with step 5 .

5 Examine the contents of the remaining registers with the exception of registers 7, 11, and 12 whose contents are fixed. By referring to the compiler map that was produced when the VS BASIC processor was installed, determine if any of the compile-time registers point to a statement processing routine. If so, that routine was processing at the time of the error. This will indicate the type of statement involved.

6 Compare the object code generated with the pseudo-assembler code skeletons shown in Appendix A. At this point, you must determine the nature of the problem, design a method for coding around it, and obtain information for submitting an APAR. (See the information on APARs in this section.) This completes the error diagnosis that is possible for compilations.

7 Examine the contents of register 14 (RP14). During execution, register 14 usually points to the instruction that was executing when the failure occurred. There are two possibilities. The instruction is in the object code of your program or in the code of a VS BASIC run-time library routine that was called by your program. To determine which is the case, compare the contents of register 14 with the contents of BSOBJ in PRG. If the value in register 14 is higher than the value in BSOBJ, the failure occurred in a program statement. If the value is lower the failure occurred in a library routine. For failures in a library routine, continue with step 8; for failures in a program statement, go to step 9 and continue.

8 Locate the library routine that was executing at the time of the failure. Refer to the library map that was produced when the VS BASIC Processor was installed. Compare the contents of register 14 with the starting addresses of the library routines shown in the map. The library routine that contains the address in register 14 was executing at the time of the failure. At this point you must determine the nature of the problem in the library routine, design a method for coding around it, and obtain information for submitting an APAR. (Refer to the information on APARs in this section.) This completes the error diagnosis that is possible for a library routine during execution.

9 Locate the failing statement. Subtract the value in BSOBJ from the value in register 14. BSOBJ is located at a displacement of A80 hex from the beginning of PRG. This is the offset of the instruction that failed. To determine which statement contains this instruction, use the line pointers table (LINPTRS). This table lists the offset of the beginning of each statement in the program together with the line number of the statement. Compare the instruction offset with the offsets listed in the line pointers table. The statement that contains the offset was executing at the time of the failure. At this point you must determine the nature of the problem in the statement, design a method for coding around it, and obtain information for submitting an APAR. This completes the error diagnosis that is possible for a program statement during execution.

INFORMATION NEEDED FOR APARS

Refer to the publication Field Engineering Programming System Language and Sort Processors Abstract Guide. It contains the information required for a PASS search argument and APAR abstract. The following information is required for submitting an APAR.

1 Component ID and Release level 5847-xx1. The items are printed at the top of the first page of the VS BASIC listing.

| 2 Operating System. The VSBSID field (X'48') of the PRG communication area will contain one of these codes:

Code	System
3	CMS/CMSBATCH
2	TSO
4	OS/VS1 or OS/VS2
5	DOS/VS
6	VSPC

3 Type of Failure. Take standard action for WAIT and LOOP. Message number ICDxxx can be used to determine which module generated the error. (See table 10.) PROGCHK causes error message ICD063 or 110 with a code of 160 (if compiler) or code of 12 (if run time).

4 Time of Failure. Examine message number (ICDxxx).

Value of xxx	Component
000-199	Executor
200-399	Compiler
400-599	Run-time library
800-999	Debug

5 For Compile Time Only. Register 8, 9, or 10 will contain the base address for compiler module (ICDJxxxx). Compare against compiler map to find module in control. Location SEQCHK (X'780') contains statement number in binary of source statement being processed. Use a source listing to get statement type, for example IF, GOTO, LET.

6 Register 3, 4, 5, or 11 contains the base address of the library module in control. Compare against library map to find the module (ICDKxxxx). Register 14 points to the generated code for the statement being processed. Register 10 contains the base of generated code. The difference of these two values is the displacement into the generated code. Use LINPTRS table to find the displacement and the corresponding LINCHN table entry for the line number.

EXAMINING STORAGE DIRECTLY UNDER CMS

1 Locate the address of the entry point ICDWOBJS. Refer to the compiler map that was produced when the VS BASIC processor was installed.

2 Set a breakpoint in the program at the address of ICDWOBJS. Use the facilities of CMS DEBUG to set the breakpoint.

3 Rerun the program. If the breakpoint is not taken before the program fails, the error occurs during compilation. If the breakpoint is taken prior to failure, the error occurs during execution. If the program fails during compilation, continue with step 4. If the program fails during execution, go to step 8 and continue.

4 Display the contents of registers 1 (RP4) and 15 (RP3). During compilation, register 1 usually points to the statement that is currently being processed and register 15 points to the corresponding object code that is generated.

5 Using these registers, type out the code that they point to. If they locate the source statement and object code, go to step 7 and continue. If they do not, continue with step 6.

6 Attempt to isolate the failing statement by eliminating statements from the program, until the only remaining statement is the one in error. You can use a binary search method. Eliminate half of the statements and rerun the program. If the error still occurs, eliminate half of the remainder. Continue eliminating half the statements from the group that continues to fail until you have isolated the one you want. Care must be taken not to introduce new errors when statements are eliminated.

7 At this point, you must determine the nature of the problem and design a method for coding around the problem and obtain information for submitting an APAR. (Refer to the preceding information on APARs.) This completes the error diagnosis that is possible for compilations.

8 Display the contents of register 8. This points to the beginning of the Communications Region (PRG).

9 Display the contents of BSOBJ in PRG. It is located at a displacement of A80 hex from the beginning of PRG. This area contains the base address of the object code area (OBJAREA).

10 Display the contents of BSLINPTRS in PRG. It is located at a displacement of AA8 hex from the beginning of PRG. This is the base address of the line pointers table (LINPTRS).

11 Type out the LINPTRS table and obtain the offset of the beginning of the generated code for each statement. The table lists the line number of each statement followed by the displacement of its object code from the beginning of OBJAREA.

12 Set breakpoints at the statements in question using the address of the beginning of OBJAREA plus the displacements of the individual statements obtained from LINPTRS.

13 Trace the execution of the suspected statements, when the breakpoint is taken.

14 If you suspect that the library routine that executes a particular statement may be producing the error and not the object code, consult the library map. Locate the run-time routine to be examined and set a breakpoint at its starting address. You can now trace the execution of the library routine. This completes the error diagnosis possible during execution.

APPENDIX A: OBJECT CODE PRODUCED BY THE VS BASIC PROCESSOR

Note: Any statement marked with a double ** in Appendix A indicates the standard statement header, which in this case is:

```
          BALR   RLOC, RRPBS
```

If the TEST option is used under VM/CMS or TSO, the statement header is:

```
          L      RLINK, DBUGNTRY +4-PRG (RRPBS)
          BALR   RLOC, RLINK
```

CHAIN STATEMENT

```
CHAIN 'AAA'
```

```
**        BALR  RLOC, RRPBS                statement header
          L     RRA3,4(RRUN)               get address of chain routine
          BALR  RLOC,RRA3                  branch to it
          DC    F'1'                       number of arguments
          DC    X'090310E7'                chain program name
```

```
CHAIN B$, 'SSS SSS'
```

```
**        BALR  RLOC, RRPBS                statement header
          L     RRA3,4(RRUN)               get address of library chain routine
          BALR  RLOC,RRA3                  branch to it
          DC    F'2'                       number of arguments
          DC    X'091210F3'                chain program name
          DC    X'090710EB'                chain argument
```

CLOSE STATEMENT

```
CLOSE A$, EXIT 100
```

```
**        BALR  RLOC, RRPBS                statement header
             .
             .                             code to evaluate subscript (B$(3))
             .
             .                             code to evaluate concatenation ('A'||'B')
             .
          L     RRA5, = A (ICDKCLOS)       get library routine address
          BALR  RLOC,RRA5                  branch to library
          DC    X'09121106'                code for A$
          DC    X'00000010'                code for EXIT
          DC    X'00007FFF'                no USING clause
```

```
CLOSE FILE A$, IOERR200
```

```
**      BALR    RLOC, RRPPS             statement header
        L       RRA11,A(ICDKVCLS)       load run-time address
        BALR    RLOC,RRA11              and branch
        DC      X'091210EC'             file A$
        DC      X'00000014'             code for IOERR
        DC      X'00007FFF'             no USING clause
```

DATA STATEMENT

```
DATA 1, A, 2*3, 4*B
```

```
        DC      X'00000000'             word for current count
        DC      X'00001123'             1
        DC      X'00001126'             A
        DC      X'02000002'             2 (repetition factor)
        DC      X'00001129'             3
        DC      X'02000004'             4 (repetition factor)
        DC      X'0100112C'             B
```

DEF STATEMENT

```
DEF  FNA(B,C) = B+C
     or
DEF  FNB(D,E)
     RETURN D+E
FNEND
```

```
**  Code to Branch Around User From Expansion
        BALR  RLOC,0                    Establish using register
        L     RRA4,8(RLOC)              Get size of DEF expansion
        B     0(RRA4,ROBJ)              Branch around it
        DC    X'00000158'               Size of code

**  Code to Save Floating Point Regs
        L     RLIN BSWKTMP              Get address of work area start
        L     RRFR,BSUFUN               Develop address of work
        A     RLIN,4(RRFR)              area for this user function
        STE   FR2,0(RLIN)               Save floating register 2
        STE   FR4,4(RLIN)               Save floating register 4
        STE   FR6,8(RLIN)               Save floating register 6
        MVC   C(ONUNITS*4,RLIN),ONCELLS Save ON units
        MVI   ONCELLS,0                 Reset ON units
        MVC   ONCELLS+1((ONUNITS*4)-1), To 'SYSTEM default'
              ONCELLS

**  Code to Move an Arithmetic Argument to its Corresponding Dummy Location
        BALR  RLOC,0                    Establish using register
        LH    RRA4,2(RLINK)             Get the argument address (BDDD)
        CLI   0(RLINK),X'01'            Is it a simple arithmetic variable
        BE    A1                        If so, branch
        CLI   0(RLINK),X'04'            Is it an arithmetic array element
        BNE   ARGERR                    If not, object time error
        STH   RRA4,A2+2                 Set up following instruction
A2      L     RRA4,0                    Load array element displacement (DDDDD)
        LE    FR0,0(RRA4,RARR)          Load the argument value
        B     A3                        Branch to process
```

**See Note at the beginning of Appendix A.

```
A1      STH   RRA4,A4+2                    Set up following instruction
A4      LE    FR0,0                        Load the argument value
A3      STE   FR0,E8(RVAL1)                Store it into dummy location
        LA    RLINK,4(RLINK)               Bump to next argument
        BALR  RLOC,0
        LH    RRA4,2(RLINK)
        CLI   0(RLINK),X'01'
        BE    24(RLOC)
        CLI   0(RLINK),X'04'
        BNE   40(RLOC)
        STH   RRA4,1A(RLOC)
        L     RRA4,0                       Repeat of previous code (to move next
        LE    FR0,0(RRA4,RARR)             arithmetic argument to its corresponding
        B     2C(RLOC)                     dummy location)
        LE    FR0,0(RRA4,RARR)
        B     2C(RLOC)
        STH   RRA4,2A(RLOC)
        LE    FR0,0
        STE   FR0,EC(RVAL)
        LA    RLINK,4(RLINK)

**  Code to Save Registers RLINK and RWKSTK in the Return Stack
        L     RRA4,PSLTH                   Get current displacement in return stack
        LA    RRA5,8(RRA4)                 Advance to next location in return stack
        LR    RLOC,RLINK                   Set up RLOC to give good error message if
                                             needed
        C     RRA5,PSMAX                   Has return stack overflowed
        BNL   STKERR                       If so, branch for error
        A     RRA4,PSTPTR                  Form return stack address
        ST    RLINK,0(RRA4,RRPBS)          save RLINK register
        ST    RWKSP,4(RRA4,RRPBS)          save RWKSP register
        LA    RRA11,0                      Get code byte to indicate user
        STC   RRA11,4(RRA4,RRPBS)          Insert code byte in return statement
        ST    RRA5,PSLTH                   Save return stack pointer
        LR    RWKSP,RLIN                   Place new workspace address in RWKSP
        L     RLIN,BSLNPTRS                Reload RLIN register

**  Code to place result in floating register 0 and restore floating point registers
        LE    FR0,E8(RVAL1)                Load B
        AE    FR0,EC(RVAL1)                Add C; Leave result in floating register 0
|       MVC   ONCELLS(ONUNITS*4),C(RWKSP)  Restore ON units
        LE    FR2,0(RWKSP)                 Restore floating register 2
        LE    FR4,4(RWKSP)                 Restore floating register 6
        LE    FR6,8(RWKSP)                 Restore floating register 6

**  Code to restore registers RLINK and RWVSP and return to caller
        L     RRA4,PSLTH                   Get displacement pointer to return stack
        SH    RRA4,82E(RRPBS)              Position it down to this entry
        ST    RRA4,PSLTH                   Save new position
        LA    RWKSP,0(RRA4,RRPBS)          Form a complete address
        A     RWKSP,PSPTR                  to return stack entry
        CLI   4(RWKSP),X'00'               Was last entry a user function entry
        BNE   RETURNER                     If not, object time error
        L     RLINK,0(RWKSP)               Restore caller's RLINK
        L     RWKSP,4(RWKSP)               Restore callers RWKSP
        BCR   15,RLINK                     Branch back to caller
        NOPR
```

| **See Note at the beginning of Appendix A.

DELETE FILE STATEMENT

```
DELETE FILE  A$, KEY = B$
```

```
**      BALR   RLOC,RRPBS                statement header
        L      RRA11,A(ICDKVDEL)         Load run-time address
        BALR   RLOC,RRA11                and branch
        DC     X'091210EC'               File A$
        DC     X'00007FFF'               No exit
        DC     X'00007FFF'               No USING clause
        DC     X'097E1189'               Key = B$
```

END STATEMENT

```
END RC=4
```

```
**      BALR   RLOC,RRPBS                statement header
        LE     FR0,20(RVAL1)             get return code value
        L      RRA5,C4(RRUN)             get address of end routine
        BR     RRA5                      go to it
```

**See Note at the beginning of Appendix A.

```
FOR I = J TO K STEP N   NEXT I
```

```
**      BALR  RLOC,RRPBS              statement header
        LE    FR0,10C(RVAL1)          get J value
        STE   FR0,7A8(RRPBS)          put it in a temp
        LE    FR0,110(RVAL1)          get K value
        STE   FR0,114(RVAL1)          put it in a temp
        LE    FR0,118(RVAL1)          get N value
        STE   FR0,11C(RVAL1)          put it in a temp
        LTER  FR0,FR0                 check N value for positive or negative
        BALR  RLOC,0                  to address FOR code
        LH    RRA5,828(RRPBS)         get a LTER instruction
        BNM   C(RLOC)                 if N not negative skip next instruction
        LH    RRA5,82A(RRPBS)         get a LCER if N is negative
EFPOS   STH   RRA5,28(RLOC)           put instruction in OVERLAY location
        LE    FR0,7A8(RRPBS)          reload J value in register
        B     20(RLOC)                skip over next time code
NEXTIME LE    FR0,108(RVAL1)          for second time around get current I value
        AE    FR0,11C(RVAL1)          and increment by N value
SKIPINIT LER  FR2,FR0                 get current loop value in register 2
        SE    FR2,114(RVAL1)          subtract the K value
        BALR  RRFR,0                  set temporary base
OVERLAY LPR   RRFR,RRFR               overlaid instruction
        BNH   16(RRFR)                loop not complete
        MVI   0(RRFR),X'10'           set for loop inactive
        MVI   1(RRFR),X'22'           by creating LPR 2,2 instruction
        L     RRA5,BCD(RRPBS)         get the past FOR address
        B     0(RRA5,ROBJ)            branch to it - loop done
SETNEWI STE   FR0,108(RVAL1)          set new value for I
NEXT
**      BALR  RLOC,RRPBS              statement header
        L     RRA5,00C(RLIN)          get offset to FOR statement from LINPTRS
        LA    RRA5,36(RRA5)           bump to NEXTIME offset
        BAL   RLOC,0(RRA5,ROBJ)       go to NEXTIME code in FOR-loop
```

FORM STATEMENT

```
FORM 3*C5,SKIP2, POS3, 5 X 4, L, NC7.3, PD8.4, PIC(B$,###.##$)
```

```
        DC    X'0100'                 Form header
        DC    X'80BDDD'               repetition factor (3*)
        DC    X'04BDDD'               C
        DC    X'01BDDD'               SKIP
        DC    X'02BDDD'               POS
        DC    X'06'                   S
        DC    X'03BDDD'               X
        DC    X'07'                   L
        DC    X'400703'               NC 7.3
        DC    X'200F04'               PD 8.4
        DC    X'08 02 09 80 06 02 01 08'  PIC header information
        DC    X'216B2020204b2020'     PIC pattern
```

Note "BDDD" represents a 2-byte base displacement field, where B is the base register and DDD is the displacement off B.

**See Note at the beginning of Appendix A.

GET STATEMENTS

```
GET A$, A, A$, B(3), B$(3), MATC, MATC8
```

```
**      BALR  RLOC,RRPBS                statement header
        L     RRA5,=A(ICDKGET)          get library routine address
        BALR  PROC,RRA5                 branch to library
        DC    X'09121106'               code for A$ (file name)
        DC    X'00007FFF'               word for error exits (7FFF' indicates name)
        DC    X'01001138'               code for A
        DC    X'09001100'               CODE FOR A$
        DC    X'40000000'               call-me code to branch to generated code
                .
                .                       code to evaluate B(3) subscript
                .
        DC    X'04000000'               code for B(3)
        DC    X'90000000'               call-me code to branch to generated code
                                        code to evaluate B$(3) subscript

        DC    X'0C12C004'               code for B$(3)
        DC    X'02000004'               code for MATC
        DC    X'0A00003E'               code for MATC8
        DC    X'FFFF'                   end-of-list indicator
```

**See Note at the beginning of Appendix A.

```
GET A$, D, D$, EXIT 100
```

```
**      BALR   RLOC,RRPBS              statement header
        L      RRA5,=A(ICDKGET)        get library routine base
        BALR   RLOC,RRA5               branch to library
        CD     X'09121106'             code for A$ (file name)
        DC     X'00000000'             code word for EXIT
        DC     X'010010E8'             code for D
        DC     X'090010EC'             code for D$
        DC     X'FFFF'                 end-of-list indicator
```

```
GET A$, D, D$, EOF 10, CONV20, IOERROR 30
```

```
**      BALR   RLOC,RRPBS              statement header:
        L      RRA5,=A(ICDKGET)        get library routine base
        BALR   RLOC,RRA5               branch to library
        DC     X'09121106'             code for A$ (file name)
        DC     X'00000004'             code for error processing (EOF CONV IOERR)
        DC     X'010010E8'             code for D
        DC     X'090010EC'             code for D$
        DC     X'FFFF'                 end-of-list indicator
```

GOSUB STATEMENT

```
GOSUB 500
```

```
**      BALR   RLOC,RRPBS              statement header
        L      RRA5,50(RRPBS)          get offset of current entry in return stack
        LA     RRFR,0(RRPBS,RRA5)      add offset to PRG address
        A      RRFR,4C(RRPBS)          and add on offset of base of return stack
        LA     RRA5,4(RRA5)            bump R5 to next entry in stack
        C      RRA5,760(RRPBS)         are there too many entries?
        BNL    24(ROBJ)                yes, branch to error routine
        ST     RRA5,50(RRPBS)          set R5 for the next stack entry
        LA     RLINK,48(RLOC)          set R15 to statement following this one
        ST     RLINK,0(RRFR)           save that address in RETURN stack
        MVI    0(RRFR),X'01'           indicate GOSUB return
        L      RRA5,20(RLIN)           get offset for statement 500 from LINEPTRS
                                         table
        B      0(ROBJ,RRA5)            branch to it.
        MVI    0(RRFR),X'01'           set the GOSUB entry indicator
        B      0(RRA5,ROBJ)            branch to the proper statement number
COUNT   DC     X'46 000003'            count of number of statement numbers
BASE    DC     X'00000000'             save area for LOAD table base
NEXT    EQU    *                       address of next statement
```

**See Note at the beginning of Appendix A.

```
GOSUB 350, 360, 370 ON X+Y
```

```
**      BALR   RLOC,RRPBS          statement header
        ST     RLOC,114(RLOC)      save LOAD table base in BASE
        B      20(RLOC)            branch around LOAD table to PAST
LOAD    L      RRA5,1C(RLIN)       load offset from BSOBJ for 350
        L      RRA5,20(RLIN)       load offset from BSOBJ for 360
        L      RRA5,24(RLIN)       load offset from BSOBJ for 370
PAST    LE     FR0,120(RVAL1)      get value of X
        AE     FR0,124(RVAL1)      add on value of Y
        BALR   RRFR,0              use R2 as gosub code base
        L      RLOC,84(RRFR)       restore R14 with LOAD table base from BASE
        LA     RLINK,88(RRFR)      set R15 to NEXT statement code
        AU     FR0,7F8(RRPBS)      make x+y expression into an integer
        BNPR   RLINK               if not positive, go to NEXT statement
        STE    FR0,85C(RRPBS)      save expression value
        CE     FR0,050(RRFR)       compare it to COUNT of statement numbers
        BHR    RLINK               if high, go to NEXT statement
        LH     RRA5,85E(RRPBS)     get expression value
        SLL    RRA5,2              multiply by 4 to index LOAD table
        LA     RRA5,4(RRA5)        add on 4 for leading ST and B instructions
        EX     RRA0,0(RRA5,RLOC)   execute the proper load instruction for
                                     statement offset
        L      RAA4,50(RRPBS)      get offset of current entry in Return stack
        LA     RRFR,0(RRPBS,RRA4)  add offset to prog address
        A      RRFR,4C(RRPBS)      and add on offset of base of stack
        LA     RRA4,4(RRA4)        bump R4 to next entry in stack
        C      RRA4,760(RRPBS)     are there too many entries?
        BNL    24(ROBJ)            yes, branch to the error processor
        ST     RRA4,50(RRPBS)      set R4 for the next stack entry
        ST     RLINK,0(RRFR)       set NEXT statement as return address in the
                                   return stack
```

GO TO STATEMENT

```
GO TO 600
```

```
**      BALR   RLOC,RRPBS          statement header
        L      RRA5,30(RLIN)       get offset to statement 600 from LINEPTR
                                   table
        B      0(RRA5,ROBJ)        branch to statement 600
```

```
GOTO 450, 550, 650 ON A**B
```

```
**      BALR   RLOC,RRPBS          statement header
        ST     RLOC,86(RLOC)       save base register in BASE
        B      20(RLOC)            branch to PAST around LOAD table
        L      RRA5,38(RLIN)       load offset to statement 450
        L      RRA5,3C(RLIN)       load offset to statement 550
        L      RRA5,40(RLIN)       load offset to statement 650
PAST    LE     FR2,12C(RVAL1)      set floating register 2 with B
        LE     FR2,128(RVAL1)      set floating register 0 with A
        L      RRA11,150(RRUN)     get address of exponentiation routine
        BALR   RLOC,RRA11          go to it.
        NOPR
        BALR   RLINK,0             use R15 as base for code
        L      RLOC,48(RLINK)      get value in BASE for R14 for LOAD table base
        AU     FR0,7F8(RRPBS)      make expression result an integer
```

**See Note at the beginning of Appendix A.

```
         BNP   52(RLINK)          if not positive, branch to NEXT statement
         STE   FR0,85C(RRPBS)     save expression result
         CE    FR0,44(RLINK)      compare with COUNT of statement number
         BH    52(RLINK)          if more, go to NEXT statement
         LH    RRA5,85E(RRPBS)    otherwise, get expression value
         SLL   RRA5,2             multiply by 4 to index into LOAD table
         LA    RRA5,4(RRA5)       add on 4 to take care of ST and B before the
                                    table
         EX    0(RRA5,RLOC)       execute the proper load of statement offset
COUNT    B     0(RRA5,ROBJ)       and branch to the proper statement
         DC    X'46 000003'       COUNT of number of statement numbers
BASE     DC    X'00 00 00 00'     save area of LOAD table base
NEXT     EQU   *
```

IF STATEMENT

```
IF B$ = '234' GOTO 800
```

```
**       BALR  RLOC,RRPBS         statement header
         LA    ROBJ,0F4(RVAL1)    get address of B$ in R10
         LA    RRFR,131(RVAL1)    get address of '234' in R2
         L     RRA11,800(RRPBS)   put pad character in R11
         IC    RRA11,0F3(RVAL1)   put length of B$ in R11
         SR    RRA3,RRA3          zero out R3
         IC    RRA3,130(RVAL1)    put length of '234' in R3
         CLCL  RRFR,ROBJ          compare B$ with '234'
         L     ROBJ,A80(RRPBS)    restore R10 with BSOBJ
         L     RRA5,48(RLIN)      get offset of statement 800 from LINEPTRS
                                    table
         BE    0,(ROBJ,RRA5)      branch to statement 800 if equal.
```

```
IF A + B >= 100 THEN A = A + 1
```

```
**       BALR  RLOC,RRPBS         statement header
         LE    FR0,128(RVAL1)     get variable A
         AE    FR0,12C(RVAL1)     add on B
         STE   FR0,0(RWKSP)       Put result in workspace
         LE    FR0,134(RVAL1)     get constant 100
         CE    FR0,0(RWKSP)       compare it to workspace value
         BALR  RLOC,0             for addressability of constant
         BNL   16(RLOC)           branch to THEN if equal or greater
         L     RRA5,12(RLOC)      get offset of code past THEN
         B     0(ROBJ,RRA5)       branch past the THEN code
         DC    X'00 00 02 6E'     offset off code past the THEN
THEN     BALR  RLOC,0             THEN statement header
         LE    FR0,128(RVAL1)     get variable A
         AE    FR0,8(RVAL1)       add on constant 1
         STE   FR0,4(RWKSP)       save value in workspace
         LE    FR0,4(RWKSP)       get workspace value
         STE   FR0,128(RVAL1)     store it in A
PAST     EQU   *
```

**See Note at the beginning of Appendix A.

```
IF L$ <> M$ THEN STOP ELSE L$ = 'FLOWER'
```

```
**      BALR  RLOC,RRPBS          statement header
        LA    ROBJ,140(RVAL1)     set R10 to address of L$
        LA    RRFR,153(RVAL1)     set R2 to address of M$
        L     RRA11,800(RRPBS)    set R11 to pad character
        IC    RRA11,13F(RVAL1)    get length of L$ into R11
        SR    RRA3,RRA3           zero out R3
        IC    RRA3,152(RVAL1)     get length of M$ into R3
        CLCL  RRFR,ROBJ           compare long
        L     ROBJ,A80(RRPBS)     reset R10 to BSOBJ
        BALR  RLOC,0              for addressability of constant
        BNE   16(RLOC)            branch to THEN clause if not equal
        L     RRA5,12(RLOC)       pick up offset from BSOBJ to ELSE
        B     0(ROBJ,RRA5)        branch to ELSE clause
        DC    X'000002B8'         offset of ELSE clause
THEN    BALR  RLOC,0              statement header for THEN clause
        L     RRA3,C4(RRUN)       pick up address of STOP routine
        BR    RRA3                branch to it
        BALR  RLOC,0              for addressability of constant
        L     RRA5,8(RLOC)        pick up offset of code past ELSE clause
        B     0(ROBJ,RRA5)        branch to it
        DC    X'000002DE'         offset past ELSE clause
ELSE    BALR  RLOC,0              statement header for ELSE clause
        LA    ROBJ,138(RVAL1)     set R10 to address of 'FLOWER'
        L     RRA11,800(RRPBS)    set R11 with pad character
        IC    RRA11,0(ROBJ)       set R11 with length of 'FLOWER'
        LA    ROBJ,1(ROBJ)        set R10 past the length byte
        LA    RRFR,13F(RVAL1)     set R2 to L$
        XR    RRA3,RRA3           zero out R3
        IC    RRA3,0(RRFR)        set R3 with length of L$
        LA    RRFR,1(RRFR)        bump R2 past the length byte
        MVCL  RRFR,ROBJ           move 'FLOWER' into L$
        L     ROBJ,A80(RRPBS)     restore R10 to BSOBJ
PAST    EQU   *                   end of ELSE clause
```

IMAGE STATEMENT

```
: AVERAGE IS ## ##
```

```
        DC    X'0200'             Image statement header
        DC    X'050B'             Literal information header
        DC    X'C1E5C5D9'
              X'C1C7C540'         Literal data
              X'C9E240'
        DC    X'10020800040205'   Conversion information
```

```
INPUT A,A$,B(3),B$(3),MAT C,MAT C$
```

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA5,=A(ICDKINPT)   get library routine address
        BALR  RLOC,RRA5           branch to library
        DC    X'01 00 1 0E8'      code for A
```

**See Note at the beginning of Appendix A.

```
         DC    X'090010EC'          code for A$
         DC    X'40000000'          call-me code to branch to generated code
                  .
                  .
                  .                 code to evaluate and check the B(3) subscript
                  .                 and put information in the WORKSPACE area
                  .
                  .
         L     RRA3,KROUTRET        get address of return point in library
         BALR  RLOC,RRA3            return to library
         DC    X'0400C000'          code for B(3)
         DC    X'40000000'          call-me code again
                  .
                  .
                  .                 generated code for B$(3) subscript
                  .                 evaluation again
                  .
                  .
         DC    X'0C12C004'          code for B$(3)
         DC    X'02000004'          code for MAT C
         DC    X'0A00003E'          code for MAT C$
         DC    X'FFFF'              end of list indicator
```

INPUT FROM 'ABC'

```
**       BALR  RLOC,RRPBS           statement header
         L     RRA5,IB0(RRUN)       get library routine address
         BALR  RLOC,RRA5            go to it
         DC    X'0901112C'          code for 'ABC' and input
```

** See Note at the beginning of Appendix A.

LET STATEMENT

```
LET A, B, C = F + A * R
```

```
**      BALR   RLOC,RRPBS              statement header
        LE     FR0,128(RVAL1)          get 'A'
        ME     FR0,170(RVAL1)          multiple by 'R'
        AE     FR0,16C(RVAL1)          add on 'F'
        STE    FR0,0(RWKSP)            save result in workspace
        LE     FR0,0(RWKSP)            get evaluated result
        STE    FR0,128(RVAL1)          store it in 'A'
        LE     FR0,0(RWKSP)            get evaluated result
        STE    FR0,12C(RVAL1)          store it in 'B'
        LE     FR0,0(RWKSP)            get evaluated result
        STE    FR0,174(RVAL1)          store it in 'C'
```

```
LET A = 67
```

```
**      BALR   RLOC,RRPBS              statement header
        LE     FR0,168(RVAL1)          get value '67'
        STE    FR0,128(RVAL1)          store it in 'A'
```

```
LET M$ = 'LL' 11 B$
```

```
**      BALR   RLOC,RRPBS              statement header
        MVC    0(3,RWKSP),178(RVAL1)   move 'LL' into workspace
        MVC    3(17,RWKSP),F4(RVAL1)   move B$ into WKSP after 'LL'
        MVI    0(RWKSP),20             set string length to 20
        LA     ROBJ,0(RWKSP)           get string address in R10
        L      RRA11,800(RRPBS)        get pad character in R11
        IC     RRA11,0(ROBJ)           get string length in R11 too
        LA     ROBJ,1(ROBJ)            bump R10 past string length
        LA     RRFR,152(RVAL1)         set target address in R2
        XR     RRA3,RRA3               zero R3
        IC     RRA3,0(RRFR)            get target length in R3
        LA     RRFR,1(RRFR)            bump R2 past target length
        MVCL   RRFR,ROBJ               move string to target
        L      ROBJ,A80(RRPBS)         restore R10 to BSOBJ
```

MAT STATEMENT

```
MAT H = (501*N-34.9) *I
```

```
**      BALR   RLOC,RRPBS              statement header
        LE     FR0,336(RRA0,RVAL1)     get constant 501
        ME     FR0,252(RRA0,RVAL1)     multiply by N
        SE     FR0,340(RRA0,RVAL1)     subtract constant 34.9
        L      RAA5,124(RRUN,RRA0)     get address of MAT scalar times array routine
        BALR   RLOC,RAA5               branch to it
        DC     X'02 00 000E'           parameter H
        DC     X'02 00 0010'           parameter I
```

**See Note at the beginning of Appendix A.

MAT E = F * G

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA5,120(RRUN,RRA0) get address of MAT mult routine
        BALR  RLOC,RRA5           branch to it
        DC    X'02 00 0008'       parameter E
        DC    X'02 00 000A'       parameter F
        DC    X'02 00 000C'       parameter G
```

MAT E = F - G

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA5,128(RRUN,RRA0) get address of MAT MINUS routine
        BALR  RLOC,RRA5           branch to it
        DC    X'02 00 0008'       parameter E
        DC    X'02 00 000A'       parameter F
        DC    X'02 00 000C'       parameter G
```

MAT E = F + G

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA5,92(RRUN,RRA0)  get address of MAT plus routine
        BALR  RLOC,RRA5           branch to it
        DC    X'02 00 0008'       parameter E
        DC    X'02 00 000A'       parameter F
        DC    X'02 00 000C'       parameter G
```

MAT C = D

```
**      BALR  RLOC,RRPBS          statement header
        BCR   0,0                 no-op
        L     RRA5,104(RRUN,RRA0) get address of MAT ASSIGN routine
        BALR  RLOC,RRA5           branch to it
        DC    X'02 00 0004'       parameter C
        DC    X'02 00 0006'       parameter D
```

MAT A = (22.7*V+3)

```
**      BALR  RLOC,RRPBS          statement header
        BCR   0,0                 no-op
        LE    FR0,244(RRA0,RVAL1) get constant 22.7
        ME    FR0,248(RRA0,RVAL1) multiply by V
        AE    FR0,24(RRA0,RVAL1)  add on 3
        L     RRA5,124(RRUN,RRA0) get address of MAT scalar assign routine
        BALR  RLOC,RRA5           branch to it
        DC    X'02 00 0000'       parameter A
        DC    X'00 00 0000'       not used, (only used for character
                                  assignment)
```

**See Note at the beginning of Appendix A.

```
MAT A(N,M) = (22.7*Y+3)
```

```
**      BALR   RLOC,RRPBS                statement header
        LE     FR0,252(RRA0,RVAL1)       get variable N
        STE    FR0,2128(RRA0,RRPBS)      save it in Redim1
        LE     FR0,256(RRA0,RVAL1)       get variable M
        STE    FR0,2132(RRA0,RRPBS)      save it in Redim2
        L      RRA5,176(RRUN,RRA0)       get address of Redim Routine
        BALR   RLOC,RRA5                 branch to it
        DC     X'02 00 0000'             parameter 'A' for redimensioning

        LE     FR0,260(RRA0,RVAL1)       get constant 22.7
        ME     FR0,248(RRA0,RVAL1)       multiply by Y
        AE     FR0,24(RRA0,RVAL1)        add on 3
        L      RRA5,124(RRUN,RRA0)       get address of scalar assign routine
        BALR   RLOC,RRA5                 branch to it
        DC     X'02 00 0000'             parameter 'A'
        DC     X'00 00 0000'             not used, (only used for character
                                         assignment)
```

```
ON OFLOW GOTO 350
```

```
        BALR   R14,R18                   statement header
        L      R5,=A(ICDKON)             get library routine @
        BALR   R14,R5                    and branch to it
        DC     X'03 04 0018'             disp into ONCELS=03
                                         condition is GOTO=04
                                         disp into LINEPTRS=0018
```

OPEN STATEMENT

```
OPEN A$ OUT,'FILEA' IN, IOERR 300
```

```
**      BALR   RLOC,RRPBS                statement header
        L      RRA5,=A(ICDKOPEN)
        BALR   RLOC,RRA5
        DC     X'09 01 1106'             code for A$ OUT
        DC     X'00 00 0008'             code for IOERR
        DC     X'00 00 7FFF'             no USING clause
        L      RRA5,=A(ICDKOPEN)
        BALR   RLOC,RRA5
        DC     X'09 00 10FF'             code for 'FILEA' IN
        DC     X'0000 0008'              code for IOERR
        DC     X'0000 7FFF'              no USING clause
```

OPEN FILE STATEMENT

```
OPEN FILE A$ IN, 'FILE1' OUT REUSE, B$(1) ALL, 'A'||'B' ALL HOLD REUSE, EXIT 50
```

```
**      BALR   RLOC,RRPBS                statement header
        L      RRA11,=A(ICDKVOPN)        get library routine
        BALR   RLOC,RRA11                branch to library
        DC     X'09 00 1106'             code for A$ IN
        DC     X'0000 0004'              code for EXIT
        DC     X'0000 7FFF'              no USING clause
        L      RRA11,=A(ICDKVOPN)        get library routine
        BALR   RLOC,RRA11                branch to library
        DC     X'09 04 10FF'             code for 'FILE1' OUT REUSE
```

**See Note at the beginning of Appendix A.

```
        DC    X'0000 0004'              code for EXIT
        DC    X'0000 7FFF'              no USING clause
                 .                      code to evaluate subscript B$(1)
                 .
                 .
        L     RRA11,=A(ICDKVOPN)        get library routine
        BALR  RLOC,RRA11                branch to library
        DC    X'00 02 C000'             code for B$(1) ALL
        DC    X'00 00 0004'             code for EXIT
        DC    X'00 00 7FFF'             no USING clause
                 .
                 .
                 .                      code to concatenate 'A'||'B'
                 .
                 .
        L     RRA11,=A(ICDKVOPN)        get library routine
        BALR  RLOC,RRA11                branch to library
        DC    X'09 01 C004'             code for 'A'||'B' ALL HOLD REUSE
        DC    X'00 00 0004'             code for EXIT
        DC    X'0000 7FFF'              no USING clause
```

PAUSE STATEMENT

| no PAUSE |

```
        BALR  RLOC,RRPBS                statement header
        L     RRA5,RRA4(RRUN)           get address of input routine
        BALR                            branch to it
        DC    X'02 F1F7F04040'          argument.  Leading '02' indicates pause, the
                                        statement number follows in EBCDIC.
```

*note that input library routine also handle run-time pause processing

PRINT STATEMENT

| PRINT A; A$, B(3); B$(3), MAT C; MAT C$ |

```
**      BALR  RLOC,RRPBS                statement header
                 .
                 .                      code to evaluate and check the B$(3)
                 .                      subscript and put information in workspace
                 .                      area.
                 .
        L     RRA5,=A(ICDKPRNT)         get library address
        BALR  RLOC,RRA5                 go to library
        DC    X'00 000000'              unformatted print indicator
        DC    X'01 02 10E8'             code for A with semi-colon following
        DC    X'09 01 1 0EC'            code for A$ with comma following
        DC    X'04 02 C000'             code for B(3) with semi-colon following
        DC    X'0C 01 C 004'            code for B$(3) with comma following
        DC    X'02 02 0 004'            code for MAT C with semi-colon following
        DC    X'0A 08 0 034'            code for MAT C$ with C/R following
        DC    X'FFFF'                   end of list indicator
```

**See Note at the beginning of Appendix A.

```
PRINT TO A$

**      BALR  RLOC,RRPBS          statement header
        L     RRA5,1B0(RRUN)      get library routine address
        BALR  RLOC,RRA5           go to it
        DC    X'09 00 1432'       code for A$ and output
```

```
PRINT USING 100, D, D$

**      BALR  RLOC,RRPBS          statement header
        L     RRA5,=A(ICDKPRNT)   get library routine address
        BALR  RLOC,RRA5           branch to library
        DC    X'01000000'         formatted print header
        DC    X'01 02 10 E8'      code for D
        DC    X'090810 EC'        code for D$
        DC    X'FFFF'             end of list indicator
```

PUT STATEMENT

```
PUT A$, A, A$, B(3), B$(3), MATC, MAT C$

**      BALR  RLOC,RRPBS          statement header
          .
          .                       code to evaluate B(3) subscript
          .
          .                       code to evaluate B$(3) subscript
          .
        L     RRA5,=A(ICDRPUT)    get library routine base
        BALR  RLOC,RRA5           branch to library
        DC    X'09121106'         code for A$ (file name)
        DC    X'00000006'         number of arguments
        DC    X'01001138'         code for A
        DC    X'09 12 1106'       code for A$
        DC    X'0400C000'         code for B(3)
        DC    X'0C 12 C004'       code for B$(3)
        DC    X'02000004'         code for MATC
        DC    X'0A00003E'         code for MATC$
```

** See Note at the beginning of Appendix A

```
PUT A$, D, D$, EXIT 100
```

```
**      BALR  RLOC,RRPBS              statement header
        L     RRA5,=A(ICDKPUT)        get library routine address
        BALR  RLOC,RRA5               branch to library
        DC    X'09 121106'            code for A$ (file name)
        DC    X'00000003'             number of arguments
        DC    X'01001008'             code for D
        DC    X'091210EC'             code for D$
        DC    X'80000000'             code for EXIT 100
```

```
PUT A$, D, D$, EOF 100, IO ERR 300
```

```
**      BALR  RLOC,RRPBS              statement header
        L     RRA5,=A(ICDKPUT)        get library routine base
        BALR  RLOC,RRA5               branch to library
        DC    X'09121106'             code for A$ (file name)
        DC    X'0000 0003'            number of arguments
        DC    X'01001008'             code for D
        DC    X'09 12 10EC'           code for D$
        DC    X'30000008'             code for EOF100, IOERR300
```

READ STATEMENT

```
READ  A, A$, B(3), B$(3), MAT C, MAT C$
```

```
**      BALR  RLOC,RRPBS              statement header
        L     RRA5,=A(ICDK READ )     get library routine address

        BALR  RLOC,RRA5               branch to library
        DC    X'01 00 1 0E8'          code for A
        DC    X'09 00 1 0EC'          code for A$
        DC    X'40 00 0 000'          call-me code to branch to generated code
                .
                .                     code to evaluate and check the B(3) subscript
                .                     and put information in the WORKSPACE area
                .
        L     RRA3,KROUTRET           get address of return point in library
        BALR  RLOC,RRA3               return to library
        DC    X'0400 C 000'           code for B(3)
        DC    X'40 00 0 000'          call-me code again
                .
                .                     generated code for B$(3) subscript evaluation
                .                     again
                .
        DC    X'0C 12 C 004'          code for B$(3)
        DC    X'02 00 0 004'          code for MAT C
        DC    X'0A 00 0 03E'          code for MAT C$
        DC    X'FF FF'                end of list indicator
```

**See Note at the beginning of Appendix A.

READ FILE STATEMENT

```
READ FILE USING 555 Z$, KEY >= '10001', D, D$, EXIT 777
```

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA11,A(ICDKVRD)    load runtime address
        BALR  RLOC,RRA11          and branch
        DC    X'09001134'         file Z$
        DC    X'0000000C'         exit 777
        DC    X'00000004'         using form 555
        DC    X'096E117D'         key ≥ '10001'
        DC    X'01001130'         D
        DC    X'09001134'         D$
        DC    X'FFFF'             end of list
```

REREAD FILE STATEMENT

```
REREAD FILE A$, D, D$
```

```
**      BALR  RLOC,RRPBS          statement header
        L     RRA11,A(ICDKVRST)   load runtime address
        BALR  RLOC,RRA11          and branch
        DC    X'091210EC'         file A$
        DC    X'00007FFF'         no exit clause
        DC    X'00007FFF'         no using clause
        DC    X'00000000'         no key clause
        DC    X'01001130'         D
        DC    X'09001134'         D$
        DC    X'FFFF'             end of list
```

**See Note at the beginning of Appendix A.

RESET STATEMENT

```
RESET A$ END, IOERR 20

**      BALR  RLOC,RRPBS                statement header
        L     RRA5,=A(ICDKRSET)         get library routine address
        BALR  RLOC,RRA5                 branch to library
        DC    X'09 12 11 06'            code for A$ (file name)
        DC    X'00 00 0000'             code for IOERR
        DC    X'FFFF 7FFF'              indicator for END

RESET 'AA'

**      BALR  RLOC,RRPBS                statement header
        L     RRA5=,A(ICDKRSET)         get library routine address
        BALR  RLOC,RRA5                 branch to library
        DC    X'0902112F'               code for 'AA' (file name)
        DC    X'0000 7FFF'              no error exit
        DC    X'0000 7FFF'              reset to start

RESET &BUFF

**      BALR  RLOC,RRPBS                statement header
        SDR   FR0,FR0                   clear register
        STD   FR0,BUFF-VARCON(RVAL1)    store in &BUFF
```

RESET FILE STATEMENT

```
RESET FILE B$, REC=2,NOREC 30

**      BALR  RLOC,RRPBS                statement header
        L     RRA11,=A(ICDKVRST)        get library routine address
        BALR  RLOC,RRA11                branch to library
        DC    X'09 12 113A'             code for B$
        DC    X'0000 0004'              code for NOREC
        DC    X'0000 7FFF'              no USING clause
        DC    X'01 7E 1010'             code for REC=2

RESET FILE B$(3) KEY = '10005'

**      BALR  RLOC,RRPBS                statement header
              .
              .                         code to evaluate subscript
              .
        L     RRA11,=A(ICDKURST)        get library routine address
        BALR  RLOC,RRA11                branch to library
        DC    X'0C12C000'               code for B$(3)
        DC    X'0000 7FFF'              no error exits
        DC    X'0000 7FFF'              no USING clause
        DC    X'09 7E 115C'             code for KEY='10003'
```

**See Note at the beginning of Appendix A.

RESTORE STATEMENT

```
RESTORE
```

```
**      BALR   RLOC,RRPBS          statement header
        L      RPFR,BSDAT          get base of data table
        ST     RRFR,CURDAT         get in current data location
```

RETURN STATEMENT

```
RETURN
```

```
**      BALR   RLOC,RRPBS          statement header
        L      RRA3,50(RRPBS)
        SH     RRA3,82C(RRPBS)
        LA     RRFR,0(RRA3,RRPBS)
        A      RRFR,4C(RRPBS)
        TM     0(RRFR),X'FF'       is entry for gosub?
        B      32(ROBJ)            no, goto error processor
        ST     RRA3,50(RRPBS)
        L      RRFR,0(RRFR)        get return address into register 2
        BR     RRFR                go to return address
```

REWRITE FILE STATEMENT

```
REWRITE FILE USING 100 A$, KEY = '222', D, D$, EXIT 888
```

```
        BALR   RLOC,RRPBS          statement header
        L      RRA11,A(ICDKVRWR)   get runtime address
        BALR   RLOC,RRA11          and branch
        DC     X'091210EC'         file A$
        DC     X'00000014'         exit 888
        DC     X'00000000'         using form 100
        DC     X'097E11A0'         Key='222'
        DC     X'01001130'         D
        DC     X'09121134'         D$
        DC     X'FFFF'             end of list
```

**See Note at the beginning of Appendix A.

STOP STATEMENT

```
STOP
```

```
**      BALR  RLOC,RRPBS              statement header
        SER   FR0,FR0                 zero return code
        L     RRA3,C4(RRUN)           get address of stop routine
        BR    RRA3                    go to it
```

WRITE FILE STATEMENT

```
WRITE FILE USING 100 A$, D, D$, EXIT 888
```

```
**      BALR  RLOC,RRPS               base for statement
        L     RRA11,A(ICDKVWRT)       get runtime address
        BALR  RLOC,RRA11              and branch
        DC    X'091210EC'             file A$
        DC    X'00000014'             exit 888
        DC    X'00000000'             using form 100
        DC    X'00000000'             no key clause
        DC    X'01001130'             D
        DC    X'09121134'             D$
        DC    X'FFFF'                 end of list
```

**See Note at the beginning of Appendix A.

APPENDIX B: VS BASIC DEBUG INTERNAL TEXT ELEMENTS

AT SUBCOMMAND

AT Header Element

Chain Address			
06	xx	00	00
Pointer to AT Subcommand List Header Element or 0			
AT Count or 0			

xx - 04 indicates that NOTIFY is required
00 indicates that NONOTIFY is required.

AT Statement ID Element (one for each statement id or range in AT subcommand)

Chain Address			
06	xx	00	00
Statement Table Pointer (for statement-ids) or 0			
Statement Table Pointer (for statement-ids) or 0			

nn - 00 indicates the last element.
10 indicates other than last.

END SUBCOMMAND

Chain Address			
16	00	00	00
00	00	00	00
00	00	00	00

GO [TO] SUBCOMMAND

Chain Address			
0B	00	00	00
Branch Address or 0			
Program Unit Identifier or 0			

HALT SUBCOMMAND

Chain Address			
0A	00	00	00
00	00	00	00
00	00	00	00

HELP SUBCOMMAND

Chain Address			
15	00	00	00
00	00	00	00
00	00	00	00

IF SUBCOMMAND

IF Header Element

Chain Address			
0E	xx	80	80
Address of IF Condition Element			
Address of IF Subcommand List Header			

xx - 01 indicates =
02 indicates ≠ or <>
03 indicates >
04 indicates >= or ≥
05 indicates <
06 indicates <= or ≤

IF Condition Element

Chain Address			
23	Length	xx_1	xx_2
Pointer to Operand 1			
Pointer to Operand 2			

Length - is 0 if both operand 1 and operand 2 are character variables or array elements. If only one of the operands is a character variable or array element, it is the length of that operand.

$\underline{xx}_1$ - 80 indicates that another element is chained to this one for operand 1 (an IF Subscript Address Chain for array elements).
40 indicates that the variable has a negative value.
08 indicates that the variable is numeric.
01 indicates that the pointer to operand 1 is a pointer to a symbol table entry (for variables and entire arrays).
02 indicates that the pointer to operand 1 is a pointer to a constant.
03 indicates that the pointer to operand 1 is a pointer to a subscript address chain (SAC) (for array elements).

Note: These values will be combined as required.

$\underline{xx}_2$ - is the same as $\underline{xx}_1$ except that it refers to operand 2.

If a variable, an array name, a subscripted array element, or a numeric or character constant appears in an IF subcommand, the debug routines ICDISCAN and ICDTSCN are called to scan the item and produce an intermediate text element that will be incorporated into this one. See the format of the intermediate text elements in this appendix.

IF Subcommand List Header

Chain Address			
24	00	00	00
Use Count			
00			

The use count is set to 1 when the element is created. The text element for the subcommand specified with the IF subcommand is chained to this element.

IF Subscript Address Chain Element (if required)

00	00	00	00
21	00	xx	00
Pointer to Symbol Table Entry if Variable Subscript or 0			
Integer Constant to be added to Variable to Compute Subscript			

xx - 01 indicates that the positive of the variable is to be used or no variable was specified.
40 indicates that the negative of the variable is to be used (for example, x(-a)).

INTERMEDIATE TEXT ELEMENTS

00	00	00	00
00	Length or 00	00	xx
00	00	00	00 or 01
Pointer to Operand			

Length - indicates the length of character variables, array elements, or constants. The length is 0 if a numeric variable is present.

xx - is the data descriptor. See the formats of the IF, LIST, SET, or WHEN subcommands for the actual values.

Whenever a numeric or character variable, array name, subscripted array element, or constant is encountered in an IF, LIST, SET, or WHEN subcommand the routine ICDISCAN is called. ICDTSCN is called, in turn, to actually do the scanning. ICDTSCN produced the intermediate text element with the third word set to 0. When before ICDISCAN return the intermediate text element to the calling routine it sets the third word to 1. When the calling routine receives the intermediate text element, it converts it into the format shown for the IF, LIST, SET, and WHEN subcommands.

LIST SUBCOMMAND

LIST Header Element

Chain Address			
0F	00	00	00
'FF000000' (for LIST *) or 0			
Pointer to Directory Entry (for LIST *) or 0			

LIST Data Element (one required for each item in the list)

Chain Address or 0			
10	xx	xx_1	00
Pointer to $Operand_1$			
00	00	00	00

xx - 00 indicates the last element.
10 indicates other than last.

xx_1 - 80 indicates that another element is chained to this one for operand 1 (a LIST Subscript Address Chain for array elements).
40 indicates that the variable has a negative value.
08 indicates that the variable is numeric.
01 indicates that the pointer to operand 1 is a pointer to a symbol table entry (for variables and entire arrays).
02 indicates that the pointer to operand 1 is a pointer to a constant.
03 indicates that the pointer to operand 1 is a pointer to a subscript address chain (SAC) (for array elements).

Note: These values will be combined as required.

If a variable, an array name, a subscripted array element, or a numeric or character constant appears in a LIST subcommand, the debug routines ICDISCAN and ICDTSCN are called to scan the item and produce an intermediate text element that will be incorporated into this one. See the format of the intermediate text elements in this appendix.

LIST Subscript Address Chain Element (if required)

00	00	00	00
21	00	xx	00
Pointer to Symbol Table Entry if Variable Subscript or 0 Integer Constant to be added to Variable to Compute Subscript			

xx - 01 indicates that the positive of the variable is to be used or no variable was specified.
41 indicates that the negative of the variable is to be used (for example, x(-a)).

LISTBRKS SUBCOMMAND

Chain Address			
11	00	00	00
00	00	00	00
00	00	00	00

LISTFREQ SUBCOMMAND

Chain Address			
12	xy	00	00
00	00	00	00
00	00	00	00

x - 8 indicates a chain.
0 indicates the last element.

y - 2 indicates zero frequency.
0 indicates other than zero frequency.

NEXT SUBCOMMAND

Chain Address			
05	00	00	00
00	00	00	00
00	00	00	00

OFF SUBCOMMAND

Chain Address			
08	xx	00	00
Statement Table Pointer or 0			
Statement Table Pointer or 0			

xx - 00 indicates the last element.
10 indicates other than last.

OFFWHEN SUBCOMMAND

Chain Address			
26	xx	00	00
00	00	00	00
Condition ID or 0			

xx - 00 indicates last element.
10 indicates other than last.

QUALIFY SUBCOMMAND

Chain Address			
04	00	00	00
Pointer to Directory Entry or 0			
00	00	00	00

RUN SUBCOMMAND

Chain Address			
0C	00	00	00
Branch Address or 0			
Program Unit Identifier or 0			

WHEN SUBCOMMAND

WHEN Header Element

Chain Address			
C7	00	80	00
Address of Condition ID Element			
00	00	00	00

WHEN Condition ID Element

Chain Address			
22	xx	80	00
0 (if a previously defined condition is to be turned on) or Address of WHEN Condition Element			
Condition ID in EBCDIC (left justified)			

xx - 00 indicates that a previously defined condition is to be turned on.
01 indicates =
02 indicates ≠ or <>
03 indicates >
04 indicates >= or ≥
05 indicates <
06 indicates <= or ≤
08 indicates that the changes of a variable or an array element are to be monitored.

WHEN Condition Element

Chain Address			
23	Length	xx_1	xx_2
Pointer to Operand 1			
Pointer to Operand 2			

Length - is the length of the variable or array element to be monitored.

xx_1 - 80 indicates that another element is chained to this one for operand 1 (a WHEN Subscript Address Chain for array elements).
40 indicates that the variable has a negative value.
08 indicates that the variable is numeric.
01 indicates that the pointer to operand 1 is a pointer to a symbol table entry (for variables and entire arrays).
02 indicates that the pointer to operand 1 is a pointer to a constant.
03 indicates that the pointer to operand 1 is a pointer to a subscript address chain (SAC) (for array elements).

Note: These values will be combined as required.

xx_2 - is the same as xx_1 except that it refers to operand 2.

If a variable, an array name, a subscripted array element, or a numeric or character constant appears in a WHEN subcommand, the debug routines ICDISCAN and ICDTSCN are called

to scan the item and produce an intermediate text element that will be incorporated into this one. See the format of the intermediate text elements in this appendix.

WHEN Subscript Address Chain Element (if required)

00	00	00	00
21	00	xx	00
Pointer to Symbol Table Entry if Variable Subscript or 0			
Integer Constant to be added to Variable to Compute Subscript			

xx - 01 indicates that the positive of the variable is to be used or no variable was specified.
41 indicates that the negative of the variable is to be used (for example, x(-a)).

TRACE SUBCOMMAND

Chain Address			
27	xx	00	00
00	00	00	00
00	00	00	00

nn - 01 indicates that the tracing of statements is required.
10 indicates that the tracing of functions is required.

SET SUBCOMMAND

SET Header Element

Chain Address			
14	00	xx_1	xx_2
Pointer to Operand$_1$ (for the Left Side of the Expression)			
Pointer to SET Data Chain Element			

xx_1 - 80 indicates that another element is chained to this one for operand 1 (a Subscript Address Chain for array elements).
40 indicates that the variable has a negative value.
08 indicates that the variable is numeric.
01 indicates that the pointer to operand 1 is a pointer to a symbol table entry (for variables and entire arrays).
02 indicates that the pointer to operand 1 is a pointer to a constant.
03 indicates that the pointer to operand 1 is a pointer to a subscript address chain (SAC) (for array elements).

Note: These values will be combined as required.

xx - 84 indicates that a character SET Data Chain Element is present.
8C indicates that a numeric SET Data Chain Element is present.

If a variable, an array name, a subscripted array element, or a numeric or character constant appears on the left side of the expression in a SET subcommand, the debug routines ICDISCAN and ICDTSCN are called to scan the item and produce an intermediate text element that will be incorporated into this one. See the format of the intermediate text elements in this appendix.

SET Data Chain Element

Chain Address			
20	Length	00	$\underline{xx}_1$
Pointer to Operand$_2$ (for the Right Side of the Expression)			

Length - is the length of the variable or array element on the right side of the expression.

$\underline{xx}_1$ - indicates that another element is chained to this one for operand 2 (a SET Subscript Address Chain for array elements).
- 40 indicates that the variable has a negative value.
- 08 indicates that the variable is numeric.
- 01 indicates that the pointer to operand 2 is a pointer to a symbol table entry (for variables and entire arrays).
- 02 indicates that the pointer to operand 2 is a pointer to a constant.
- 03 indicates that the pointer to operand 2 is a pointer to a subscript address chain (SAC) (for array elements).

Note: These values will be combined as required.

If a variable, an array name, a subscripted array element, or a numeric or character constant appears on the right side of the expression in a SET subcommand, the debug routines ICDISCAN and ICDTSCN are called to scan the item and produce an intermediate text element that will be incorporated into this one. See the format of the intermediate text elements in this appendix.

SET Subscript Address Chain Element (if required)

00	00	00	00
21	00	xx	00
Pointer to Symbol Table Entry if Variable Subscript or 0			
Integer Constant to be added to Variable to Compute Subscript			

xx - 01 indicates that the positive of the variable is to be used or no variable was specified.
- 41 indicates that the negative of the variable is to be used (for example, x(-a)).

WHERE SUBCOMMAND

Chain Address			
13	xx	00	00
00	00	00	00
00	00	00	00

xx - 20 indicates a statement.
01 indicates a function.
00 indicates a simple message.

INDEX

VS BASIC Program Logic
LY28-6422-2

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

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 Staple

VS BASIC Program Logic Printed in U.S.A. LY28-6422-2

IBM®

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

VS BASIC Program Logic Printed in U.S.A. LY28-6422-2

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)