MUSIC/SP User's Reference Guide

Part 4 - Chapter 8 (UR_P4.PS)

Chapter 8. Processors

Overview of Processors

To run a program written in a high-level language, the system goes through several steps before the program actually starts running. The high level language input is referred to as the "source" language. The desired result of a compilation is machine language code, or *object code*, which is placed in a file known as the *object module*.

The first step is the *compile* step. Here a compiler checks over your *source* statements and the compiler may give you a *source listing* of them. You use a compiler that was specifically designed to handle the particular computer language that you are using. Any errors that the compiler finds are identified (or *flagged*) so that you can correct them. These error messages may appear within your source listing or they may appear later under a heading of *diagnostics*. The compiler produces what is known as *object code* based on your source statements and this object code is kept for use in the next step. You can get a copy of this object code if you want in a format known as an *object module*. (The object modules cannot be converted back to source programs. Thus you should not discard your source just because you now have an object module.)

The next step is the *load* step. Here the loader receives the object code produced by the compiler and loads it into main storage. Any system-supplied subroutines that your program needs are also loaded into main storage at this time.

The last step is the *execute* (GO) step. This is where the program is given control. The program communicates with the system through a set of system routines called the *user-system interface*.

Note: FSI (Full Screen Interface) provides a menu option for creating programs and using processors.

MUSIC Compilers

The MUSIC compilers automatically start the chain of events through the steps described above without you having to be aware of them. If the compiler or loader notices some errors in your program, then it stops the chain so that you can correct the errors and start again.

The name on the /LOAD statement specifies the required processor. This may be the name of a compiler, loader, interpreter, or other processor.

APL, VS APL, BASIC, and REXX operate differently from compilers and are considered *interpreters*. GDDM is a collection of subroutines and utilities used in conjunction with a compiler.

Figure 8.1 below illustrates the steps involved in processing a program through a compiler.

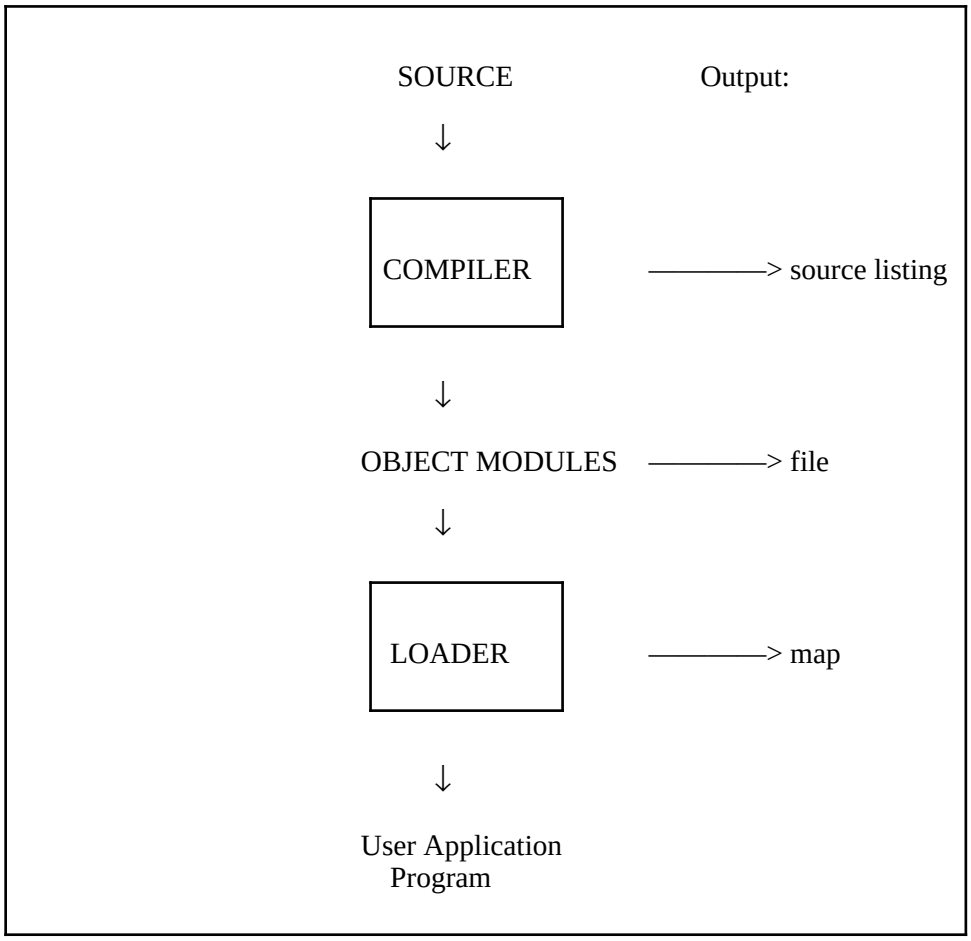

Figure 8.1 - Compilers

MUSIC Loaders

If you have object modules produced by a compiler, then you can bypass the compiler step and proceed directly to the loader step. You normally accomplish this by specifying a /LOAD ASMLG statement except in the case of PL/I. PL/I requires its own loader, /LOAD PLILG.

If parts of your program are in source form and other parts in object modules, then just use the compiler. MUSIC compilers will separate out the object modules and cause them to be loaded at the appropriate time.

MUSIC Linkage Editor

Instead of loading the object modules directly into main storage, you have the option of creating a *load module* with the MUSIC Linkage Editor and stored in a file. The MUSIC Linkage Editor is called by a /LOAD LKED. There are two main reasons why you might want to use the Linkage Editor. One is that your program is too big for all of it to fit into the MUSIC user region at one time. In this case, you use an overlay structure to split your program into segments that will each fit into the user region. The other main use of the Linkage Editor is to achieve a faster loading time for programs that are used often. You will find that it is normally faster to load a load module than loading the program from the corresponding object modules.

Figure 8.2 shows the steps involved in creating a load module with the linkage editor.

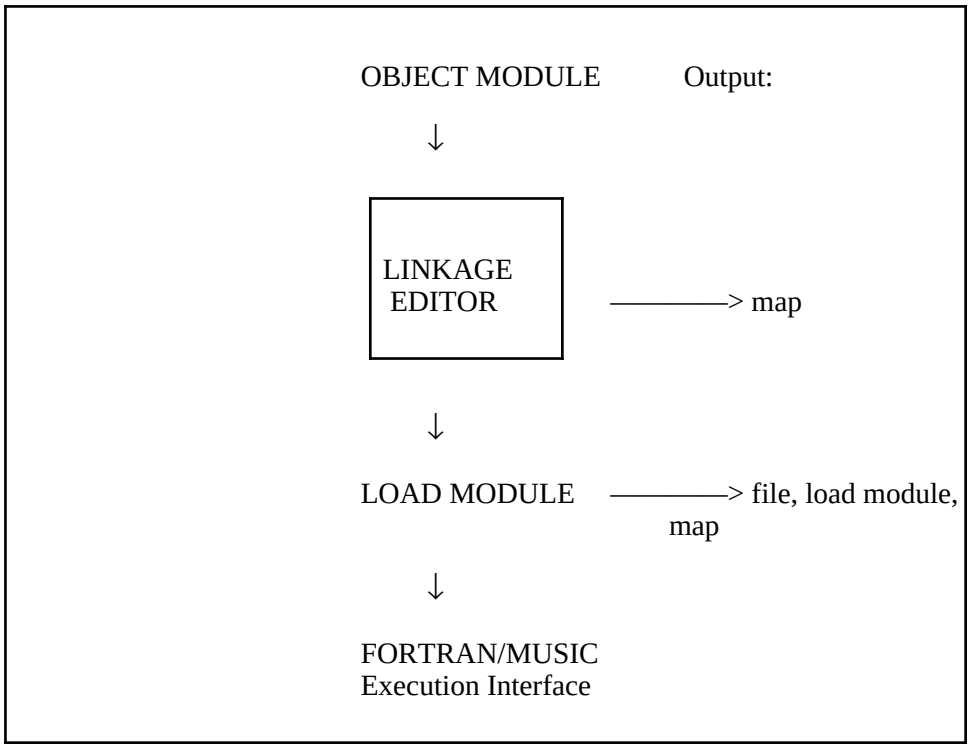

Figure 8.2 - Linkage Editor

Load Module Executors

Load module executors are used to execute a program that has been previously created with the MUSIC Linkage Editor. The loading process can be considerably faster than the use of a standard loader. The names of the two load module executors on MUSIC are: XMON and XMPLI. For a load module produced by an object module from a PL/I program, use the /LOAD XMPLI statement. You must use the /LOAD XMON statement if the load module is produced by an object module from a COBOL or VS Assembler program. The OS/MUSIC interfaces will be loaded in the last two cases.

Object modules are used to form the load modules. However, once you have a load module you cannot transform it back into object modules.

MUSIC Interpreters

An interpreter takes as input a source language. However, unlike a compiler, an interpreter does not generate machine language code in an object module. Instead, the source language statement is executed immediately. One way of thinking about it is that the machine language instructions for the current source language statement are generated, then executed, and then discarded.

Processors, such as REXX, APL, and BASIC, are interpreters, and as such do not adhere to the sequence of events described above in "MUSIC Compilers". Refer to the individual descriptions for complete information on how to run these processors on MUSIC.

Programming Tutorials

MUSIC provides a tutorial facility for learning programming languages. To invoke this facility type "TUT" in *Go mode or choose the "Tutorials" item on the FSI main menu. Help is provided once this facility is invoked.

APL - Subset Version

MUSIC supports two versions of APL. The subset APL version described below should be used only if your installation does not have VS APL (Program Product 5748-AP1).

The MUSIC/APL Subsystem is an implementation of the IBM APL S/360 Type III program. APL is a conversational time-sharing facility based on a mathematical programming language called "A Programming Language". MUSIC/APL runs under MUSIC in much the same way as any other program. The entire MUSIC/APL session is regarded by MUSIC as one job.

The APL language is concise and has a small number of syntax rules. It supports built-in mathematical operators which permit vector and matrix manipulation. It operates as both a sophisticated and powerful desk calculator and as a programming language.

Running MUSIC/APL

The user invokes MUSIC/APL by typing APL when the workstation is in command (*Go) mode. (Some installations may require you to type /EXEC APL if your user profile is set up not to allowed the implied EXEC feature.) To end the APL session, the user types the APL command)OFF. This APL command will return the user to MUSIC command mode. The APL session can also be terminated by the user typing in the MUSIC command /CANCEL.

A MUSIC execution time limit applies for your entire APL session. MUSIC/APL has a *limiting time* feature which allows user notification when excessive function evaluation time is detected. This feature is discussed in more detail later in this writeup, together with a full description of how the APL session time limit is set.

Workstation Requirement - APL

MUSIC/APL can be run from most workstations supported by MUSIC. However, it uses a different set of characters to those normally available. Some workstations have an optional APL character set feature. Operation of APL from workstations without the APL character set is not recommended. The following topics discuss the operation of APL on the different types of supported workstations.

APL on 2741 and 3767 Terminals

These APL characters are available on an IBM 2741 terminal through the use of an interchangeable type element. This element is available for both the EBCD keyboard arrangement (#1167988) and the correspondence arrangement (#1167987). The last three digits of the part number is stamped on the top rim of the type element.

The APL character set for an IBM 3767 terminal is available through the alternate character set feature.

The ATTN key on these terminals is handled according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

APL on 3270 Series Terminals

The APL character set is available on the 3270 series display terminals through the APL character set feature. To inform MUSIC that your 3270 terminal has this feature, you must use the terminal class specification of 3270A, if it is a 3277 terminal, or use 3270B, if it is a 3278 or 3279 terminal, when you sign on to MUSIC. See the /ID command for details on how to specify the trmcls parameter.

When installed, the special APL ON/OFF key is used to select the APL character set used for entry of characters. The terminal can always display APL characters. The PA1 key is used in the usual MUSIC manner to signal attention when the terminal is in WORKING mode. Escape from input in APL is through the OUT sequence signalled via the PA1 key when the terminal is in READING mode. The PA2 key is used in the usual MUSIC manner to signal that the next page is to be displayed.

APL on TTY Terminals

Some TTY terminals have an APL characters available on them. MUSIC/APL may work correctly from these devices, though you must check this out with the one you want to use. Since various manufacturers have established different conventions in APL mode, the user must indicate which one is to be used. The TRT option described below is used to specify the scheme used. (The default value of 4 suits a large number of terminals.)

```
/OPT TRT=n    (n is 1,2,3 or 4)
```

To verify that you have chosen the right number for the TRT option, execute the function KEYBOARD contained in the MUSFNS workspace. You should get a printout like the one shown in the example at the end of the VS APL discussion of this chapter of the manual.

The TTY terminals should be set to transmit even parity.

Some TTY terminals can generate a few more characters than are defined for an IBM 2741 terminal. These additional characters may be rejected if an attempt is made to enter them in APL mode.

TTY terminals have no direct equivalent of the ATTN key on an IBM 2741 terminal and so the following should be done instead: Use the LINE FEED key instead of the ATTN key to correct typing errors. Use the BREAK key instead of the ATTN key in all other cases.

Usage Notes - APL

APL workspace numbers follow a convention described under the heading "APL Workspace Concepts".

MUSIC/APL workspaces are transferable to other MUSIC/APL systems. APL workspaces generated on other systems such as OS, CMS cannot be installed on MUSIC/APL.

MUSIC/APL handles the ATTN key according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

It is advisable to periodically save your current APL workspace since its contents will be lost should your workstation become disconnected from the computer for any reason.

Each input line is checked for the MUSIC command /CAN (the abbreviation of the /CANCEL command), and if found will terminate the APL session.

APL Workspace Concepts

Your APL programs or functions are run in an area known in APL as your *workspace* (abbreviated WS). The contents of your active workspace may be saved as part of your *workspace library*. Another workspace may be loaded from this library to become the new active workspace.

The APL workspace library on MUSIC is contained on one or more User Data Set (UDS) files. The APL workspace number is used to indicate which APL workspace library is to be used. If this number is not specified, then library number 1 is assumed. The library number corresponds to the MUSIC I/O unit number as defined on the MUSIC /FILE command. The workspace size is fixed at 36,000 bytes.

Library number 3 contains the APL workspaces that are provided with the MUSIC system. These workspaces contain functions and information of common interest to MUSIC/APL users. The contents of this workspace are described later on in this writeup.

Initializing an APL WS

You can make extensive use of APL without ever having your own workspace. However, if you want to save your workspace you must set up a MUSIC UDS file to hold workspaces. The following procedure illustrates how you can initialize a MUSIC UDS file so it can be used as an APL workspace library. This procedure is done only once for each UDS file you wish to use. Each UDS file can hold many workspaces, though you will usually find that 1 or 2 is sufficient for most work. The NREC parameter on the /FILE statement should be the number of workspaces times 284. Thus NREC(568) is used to allocate space to hold 2 workspaces. A sample initializing job follows:

```
/FILE 1 UDS(dsname) NREC(568) VOL(vvvvvv) NEW
/INCLUDE APLINT
```

The *dsname* is the dsname of you file and the *vvvvvv* in this example is the disk volume that contains your workspace library. Refer to the description on the /FILE statement in *Chapter 6. MUSIC Job Control Statements* for more details about this control statement.

Using A WS Library - APL

You use the following set up to invoke MUSIC/APL with your workspace library:

```
/FILE 1 UDS(dsname) VOL(vvvvvv) OLD
/LOAD APL
```

The /FILE statement uses the same dsname and volume name that you used in the job to initialize the file. Note that the NREC parameter is not given and that the OLD is used instead of NEW.

The above set of commands can be saved on the MUSIC Save Library to make it more convenient for you to use them again at a later time. Thus, if you saved it under the name MYAPL, then all you need do is type the name MYAPL to run it at a later time.

In the above example, the workspace library has been shown as being assigned as MUSIC unit number 1. You may also use MUSIC unit numbers 2 and 3 for more APL workspace libraries. Normally you do not use unit number 3 for one of your own libraries since this will take the place of the system supplied one. You use the workspace unit number in the various APL system commands to tell APL which workspace library you want it to refer to. If not specified, then unit number 1 will be used.

APL Public Workspaces

A MUSIC/APL user can access a library of public workspaces to do various functions. This library of workspaces is accessed as library number 3. A list of all the workspace names can be obtained by the APL command)LIB 3. To find out more about a specific one simply load it into the active workspace by the APL command)LOAD 3 xxx and then type DESCRIBE.

The following is a summary of the contents of the MUSIC/APL library:

NEWS	This workspace contains information specific to APL under MUSIC such as limitations, hints and summary of the contents of the other workspaces.
MUSFNS	This workspace contains functions which when executed tell how many users are on MUSIC, how much job time you have used, etc. It also contains the function KEYBOARD that will print all the valid APL symbols.
BOATPROB	This workspace plays the age-old game of missionaries and cannibals, except this one makes sure you don't loose any missionaries by making a bad move. Furthermore, this game can be played in English, French or Spanish.
APLCOURSE	This workspace drills the user in the use of APL primitive functions and, in fact, is a good way to familiarize yourself with them.
TYPEDRILL	This workspace times how long it takes you to type lines of text which you supply. It can be used as a way to improve your typing speed.
PLOTFORMAT	This workspace contains the functions required to plot graphs on the workstation.
ADVANCEDEX	This workspace contains functions which can be used to find various mathematical numbers such as binomial coefficients, gcd's and it also contains functions to invert matrices.
WSFNS	This workspace contains functions that do the equivalent to the system functions)WIDTH,)ORIGIN, etc.
TEST2741	This workspace is useful in identifying failures of a 2741 terminal and can also be used to test it after repair. This workspace is also of some value in testing other types of supported workstations.

APL System Commands

APL uses a set of system commands to perform the various workspace functions, as well as several other activities. These APL system commands have nothing to do with MUSIC commands which may have similar names. Each APL system command starts with a ")" character in the APL character set. (Users are cautioned that this character is probably NOT the same as that used when the workstation is not using the APL character set.)

The following are the possible APL system commands that you may use:

```
)LIB      )LOAD     )SAVE     )DROP     )COPY     )PCOPY
)CLEAR    )WSID     )WIDTH    )SI       )SIV      )ORIG
)OFF      )FNS      )VARS     )ERASE    )DIGITS
```

MUSIC/APL I Beams

The APL language defines a set of functions known as "I Beam" functions. These functions can be used to obtain the current date and time as well as other information. The list below gives the I beam functions that you may use. Times are in sixtieths (1/60) of a second unless otherwise noted.

I19 Accumulated keyboard unlocked time in this APL session

I20 Time of day

I21 Accumulated processing time used in the APL session in units of 1/60 of a service unit

I22 Available space (bytes) remaining in current workspace

I23 Number of users signed on to MUSIC

I24 Time of day of the beginning of this APL session

I25 Date as a six digit integer of the form mmddyy

I26 The first element of vector I27

I27 The vector of statement numbers in the state indicator

Limiting Time Option - APL

The user can limit the amount of execution time performed between reads from the workstation. This is of particular use to stop endless loops during function evaluation.

When the time limit expires, the execution of the function will be suspended and a time exceeded message will be displayed. The specific function and line number will be identified in the same manner as if the ATTN key had been hit at that moment.

```
/OPT LIMIT=nn    (nn is time limit in service units)
```

The above control statement is used to specify the limiting time value and is placed following the /LOAD APL control statement. The actual cutoff point may be slightly greater than that specified. (Note that if other /OPT statements are used, they must each appear as separate /OPT statements as it is not permissible to combine these /OPT statements into one.)

The total amount of execution time used so far can always be determined by the I-BEAM 21 function of the APL language.

Extension of Session Time Limits - APL

The limit on the amount of execution time for a MUSIC job is normally the default job time limit associated with the user's userid. For APL, this maximum is automatically multiplied by 10 (up to a maximum of 600 service units), and a limiting time option (/OPT LIMIT=m) is set using *m* = the original time limit. This extension is not done for a batch job, or if the original time limit is 600 service units or more, or if a specific time limit is specified by /SYS TIME=n (where *n* is not MAX).

Examples:

1. If the user's default time limit is 8 SU and /SYS TIME= is not used, the APL job will get a limit of 80 SU and /OPT LIMIT=8 will be in effect.

2. If the user's default time limit is 180 SU and /SYS TIME= is not used, the APL job will get a limit of 600 SU and /OPT LIMIT=180 will be in effect.

3. If the user's maximum time limit is 16 SU and /SYS TIME=MAX is specified, the APL job will get a limit of 160 SU and /OPT LIMIT=16 will be in effect.

The user may specify a lower limiting time option if desired.

If the user specifies a job time limit via the /SYS TIME=n control statement (but not /SYS TIME=MAX), then the time extension will not be done nor will a limiting time option be set. This allows users to control the amount of processing time used in the usual manner.

References - APL

IBM APL/360 Primer (GH20-0689)

APL Interpreter - VSAPL

VS APL is an IBM Program Product (5748-AP1) that interprets statements written in APL. The APL language has a uniform notation, tailored to solving a great variety of problems interactively at a workstation. The language was originated to define problems concisely using well-known mathematical symbols. It supports built-in mathematical operators which permit efficient vector and matrix manipulation. It operates as both a sophisticated and powerful desk calculator and as a programming language.

VS APL supports a shared variable facility that allows APL users to communicate with the MUSIC files that may have been created by APL or other processors and editors running under MUSIC.

In VS APL, data and programs are gathered together and stored in user areas called *workspaces*. These workspaces can be stored for later use using APL commands. Workspaces created by other users can be accessed subject to the security rules of VS APL and MUSIC. A collection of public workspaces are available to assist users to perform various functions in APL.

The IBM publication *APL Language* (GC26-3847) should be used as the reference for APL.

Usage Notes - VSAPL

VS APL runs under MUSIC in much the same way as any other program. That is, the entire APL session is regarded by MUSIC as one job.

A MUSIC execution time limit applies for your entire APL session. VS APL has a *limiting time* feature which allows user notification when excessive function evaluation time is detected. This feature is discussed in more detail later in this writeup, together with a full description of how the APL session time limit is set.

It is advisable to periodically save your current APL workspace since its contents will be lost should your workstation become disconnected from the computer for any reason.

Running VS APL

There are two techniques to invoke VS APL. One is to type VSAPL or VSAPL.FS when the workstation is in command (*Go) mode. VSAPL.FS should only be used by users who will invoke the full screen editor functions from 3270-type workstations.

The second technique is used when you wish to specify some run options to VS APL. This is done by creating a file on the Save Library containing the sequence of commands shown below. For example, if the created file was called MYAPL, then just type the name MYAPL at *Go time to invoke VS APL with your options.

```
/SYS REGION=nnn
/FILE 1 UDS(&&TEMP)
/INCLUDE VSAPL
/OPT TRT=n
/OPT LIMIT=nnn
```

The number on the SYS command is discussed later in the topic "Workspace Concepts". The FILE command is used if the user is invoking the full-screen editor contained in the workspace called MUSIC.

The number after TRT is discussed later in the topic "APL on TTY Terminals". The number after LIMIT is discussed later in the topic "Limiting Time Option". Omit the commands that contain options you do not wish to use.

To end the APL session, the user types the APL command)OFF. This APL command will return the user to MUSIC command mode.

Workstation Requirement - VSAPL

VS APL can be run from most workstations supported by MUSIC. However, it uses a different set of characters to those normally available. Some workstations have an optional APL character set feature. Operation of APL from workstations without the APL character set is not recommended. The following topics discuss the operation of APL on the different types of supported workstations.

APL on 2741 and 3767 Terminals

These APL characters are available on an IBM 2741 terminal through the use of an interchangeable type element. This element is available for both the EBCD keyboard arrangement (#1167988) and the correspondence arrangement (#1167987). The last three digits of the part number is stamped on the top rim of the type element.

The APL character set for an IBM 3767 terminal is available through the alternate character set feature.

The ATTN key on these terminals is handled according to the usual APL rules. These rules are somewhat different to the usual MUSIC usage of this key.

APL on 3270 Series Terminals

The APL character set is available on the 3270 series display terminals through the APL character set feature. To inform MUSIC that your 3270 terminal has this feature, you must use the terminal class specification of 3270A, if it is a 3277 terminal, or use 3270B, if it is a 3278 or 3279 terminal, when you sign on to MUSIC. See the /ID command for details on how to specify the trmcls parameter.

When installed, the special APL ON/OFF key is used to select the APL character set used for entry of characters. The terminal can always display APL characters. The PA1 key is used in the usual MUSIC manner to signal attention when the terminal is in WORKING mode. Escape from input in APL is through the OUT sequence signalled via the PA1 key when the terminal is in READING mode. The PA2 key is used in the usual MUSIC manner to signal that the next page is to be displayed.

APL on TTY Terminals

Some TTY terminals have an APL characters available on them. VS APL may work correctly from these devices, though you must check this out with the one you want to use. Since various manufacturers have established different conventions in APL mode, the user must indicate which one is to be used. The TRT option described below is used to specify the scheme used. (The default value of 4 suits a large number of terminals.)

```
/OPT TRT=n    (n is 1,2,3 or 4)
```

To verify that you have chosen the right number for the TRT option, execute the function KEYBOARD contained in the MUSFNS workspace. You should get a printout like the one shown in the example at the

end of this section.

The TTY terminals should be set to transmit even parity.

Some TTY terminals can generate a few more characters than are defined for an IBM 2741 terminal. These additional characters may be rejected if an attempt is made to enter them in APL mode.

TTY terminals have no direct equivalent of the ATTN key on an IBM 2741 terminal and so the following should be done instead: Use the LINE FEED key instead of the ATTN key to correct typing errors. Use the BREAK key instead of the ATTN key in all other cases.

It is often difficult on TTY terminals to know when the system is waiting for the user to enter character data in response to a quote-quad read. The use of the /OPT BELLPR option will sound the terminal's bell when a quote-quad read is done.

APL System Commands

APL uses a set of system commands to perform the various workspace functions, as well as several other activities. These APL system commands have nothing to do with MUSIC commands which may have similar names. Each APL system command starts with a ")" character in the APL character set. (Users are cautioned that this character is probably NOT the same as that used when the workstation is not using the APL character set.)

The following are the possible VS APL system commands that you may use:

```
)CLEAR    )CONTINUE )COPY     )DROP     )ERASE    )FNS
)GROUP    )GRP      )GRPS     )LIB      )LOAD     )OFF
)PCOPY    )QUOTA    )SAVE     )SI       )SINL     )STACK
)SYMBOLS  )VARS     )WSID     )WSSIZE
```

Workspace Concepts - VSAPL

Your APL programs or functions are run in an area known in APL as your *workspace*. The contents of your active workspace may be saved as part of your *workspace library*. Another workspace may be loaded from this library to become the new active workspace.

The maximum size workspace you may use is determined by the size of the user region you are using. By default, MUSIC will use a region size of 108K which will result in a maximum workspace size of about 60K (K=1024 characters.) To use more than this, specify a larger region size on the MUSIC /SYS statement.

When saving a workspace, VS APL will write only the used part of current workspace to disk regardless of the size of the MUSIC region.

The APL workspaces on MUSIC are stored on the Save Library together with files created by other processors and editors. MUSIC automatically prefixes the workspace name with a special character sequence to make their names different to those of other files. For example, the workspace SAMPLE will be stored under the name @APLW.SAMPLE.

Workspace Compatibility - VSAPL

VS APL workspaces are transferable to other VS APL installations. See your installation support personnel for assistance in this matter. APL workspaces generated by other APL versions will not work on VS APL without conversion. It may be possible to automatically convert these workspaces. The following topic discusses how to convert workspaces created on MUSIC by the older APL version.

Workspace Conversion of old MUSIC/APL Workspaces

A conversion program exists to convert APL workspaces that were created on UDS files by the MUSIC/APL (/LOAD APL) processor. The converted workspaces will be written to the Save Library using the workspace name if possible. If the workspace name contains characters other than the standard alphanumeric ones, then the name will be converted to a suitable one. Workspace passwords, if any, will not be converted.

The conversion utility will display messages summarizing the changes made to the functions. It can run from a workstation or from batch. Running from batch may be useful when converting many workspaces.

The following sequence of commands will convert all MUSIC/APL workspaces on the UDS file described by the /FILE command. This procedure automatically requests a region size of 160K to perform the conversion.

```
/FILE 1 UDS(dsname) VOL(vvvvvv)
/INCLUDE MWSCON
   ...options if any .....
```

Use the option TEST to indicate that you do not wish the converted workspace to be written to disk. Use the option REPL option to indicate that the converted workspace is to replace any one of the same name on the user's Save Library.

Workspace Library Numbers - VSAPL

Omit the library number when referring to workspaces stored on your own library.

Use the numbers 1, 2, 3 to refer to the public workspace libraries as described in the following topic.

Use the library number 1000 to store workspaces that are to be accessible to other users. You may wish to use a workspace password to restrict access. This is equivalent of saving a public file in the MUSIC Save Library. A)LIB command will also list workspaces saved by the user using library 1000.

Use the library number 1000 to retrieve workspaces that were saved by others using library 1000. You are not permitted to perform a)LIB 1000 command.

APL Public Workspaces

The VS APL user on MUSIC can access several public workspaces libraries to do various functions. Library 1 contains workspaces that IBM distributes as part of VS APL. Library 3 contains other workspaces that are useful in the MUSIC environment. A list of all the workspace names can be obtained by the APL)LIB n command. To find out more about a specific one simply load it into the active workspace by the APL command)LOAD n xxx. Then type ABSTRACT, DESCRIBE or HOW to cause an abstract, description or

usage details to be displayed.

The following is a summary of the contents of the some of the public workspaces:

NEWS	This workspace on library 1 contains information specific to APL users.
PLOT	This workspace on library 1 contains the functions required to plot graphs on the workstation.
EXAMPLES	This workspace on library 1 contains functions that illustrate the power of APL in solving problems. Sample functions dealing with scientific, engineering, mathematical and commercial areas are presented.
FORMAT	This workspace on library 1 contains functions designed to aid in the formatting of output.
SBIC	This workspace on library 1 contains the application illustrated in the APL Language Manual (GC26-3847). It is a skeletal system for sales, billing and inventory control.
MUSFNS	This workspace on library 3 contains functions which tell how much job time you have used, the current time and date, etc. It also contains the function KEYBOARD that will print all the valid APL symbols.
BOATPROB	This workspace on library 3 plays the age-old game of missionaries and cannibals, except this one makes sure you don't loose any missionaries by making a bad move. Furthermore, this game can be played in English, French or Spanish.
TEST2741	This workspace on library 3 is useful in identifying failures of a 2741 terminal and can also be used to test it after repair. This workspace is also of some value in testing other types of supported workstations.
MUSIC	This workspace on library 3 contains a function that will invoke the MUSIC context editor to assist users of 3270 APL terminals modify their functions.

Transporting VS APL Workspaces to Other Installations

A program is available to dump (write) MUSIC VS APL workspaces to tape in a format which is transportable to VS APL installations which are not running MUSIC. The output tape is in the same format as that used by IBM when they distribute VS APL workspaces. (The FILARC program, which is described in *Chapter 10. Utilities*, should be used to send workspaces to other MUSIC installations.)

To run the program, use the following sequence of control statements:

```
/FILE 1 TAPE BLK(800) LRECL(80) VOL(xxxxxx) OLD
/INCLUDE DUMPWS
code:name1   (names of workspaces to be dumped)
code:name2
 . . .
```

The output tape must be defined by the I/O unit number 1, and must be blocked into 800 byte blocks with a logical record length of 80 bytes as shown above. Each workspace name must be specified on a separate line starting at column 1. The name must be in the form of *code*:*name*; *code* must be a valid 4 character MUSIC user code, *name* is the VS APL workspace name (maximum 11 characters). (The actual file name is "userid:@APLW.name".) The workspaces will be dumped in the order supplied and each workspace is dumped on a separate file on the tape. Since the dumped workspaces on the tape will have no workspace

names or passwords, it is advisable to keep a list of the order of the workspaces on the tape (this program produces such a list).

Limiting Time Option - VSAPL

The user can limit the amount of execution time performed between reads from the workstation. This is of particular use to stop endless loops during function evaluation.

When the time limit expires, the execution of the function will be suspended. The specific function and line number will be identified in the same manner as if the ATTN key had been hit at that moment.

```
/OPT LIMIT=nn    (nn is time limit in service units)
```

The above control statement is used to specify the limiting time value and is placed following the /LOAD VSAPL control statement. The actual cutoff point may be slightly greater than that specified.

The total amount of execution time used so far can always be determined by the "QUAD AI" APL system variable.

Extension of Session Time Limits - VSAPL

The limit on the amount of execution time for a MUSIC job is normally the default job time limit associated with the user's userid. For APL, this maximum is automatically multiplied by 10 (up to a maximum of 600 service units), and a limiting time option (/OPT LIMIT=m) is set using *m* = the original time limit. This extension is not done for a batch job, or if the original time limit is 600 service units or more, or if a specific time limit is specified by /SYS TIME=n (where *n* is not MAX).

Examples:

1. If the user's default time limit is 8 SU and /SYS TIME= is not used, the APL job will get a limit of 80 SU and /OPT LIMIT=8 will be in effect.

2. If the user's default time limit is 180 SU and /SYS TIME= is not used, the APL job will get a limit of 600 SU and /OPT LIMIT=180 will be in effect.

3. If the user's maximum time limit is 16 SU and /SYS TIME=MAX is specified, the APL job will get a limit of 160 SU and /OPT LIMIT=16 will be in effect.

The user may specify a lower limiting time option if desired.

If the user specifies a job time limit via the /SYS TIME=n control statement (but not /SYS TIME=MAX), then the time extension will not be done nor will a limiting time option be set. This allows users to control the amount of processing time used in the usual manner.

References - VSAPL

APL Language (GC26-3847)

APL/360 Primer IBM (GH20-0689)

Assembler - ASM

The MUSIC VS Assembler is the IBM OS/VS Assembler processor interfaced to run under MUSIC.

The VS Assembler programs must normally be assembled and executed using the OS/MUSIC interface. The /LOAD ASM procedure automatically loads this interface at execution time.

Assembler subroutines that do not perform input or output may be called from FORTRAN programs.

If more than one assembler module is to be compiled at the same time then they must be separated from each other by /OPT statements. If these /OPT statements are blank, then the options previously given will continue to be used. After loading is complete, the program is executed under control of the OS/MUSIC interface.

A program made up of only object modules produced by this Assembler and optionally COBOL object modules, can be loaded more quickly by using the /LOAD ASMLG procedure. Alternately the /LOAD LKED can be used to produce a load module from these modules.

Notes:

1. Since VS Assembler programs can invoke all the facilities of OS/VS operating system, some of which are not available on MUSIC, it is possible to assemble an assembler program which will not execute properly. Certain OS/VS facilities are provided by the OS/MUSIC interface. Attempts to use unavailable features results in the execution time message FEATURE NOT SUPPORTED.

2. The DXD, and CXD facilities of the VS Assembler are not supported with the /LOAD VSFORT and /LOAD LOADER procedures. Programs which are to be executed must have a CSECT or START before the first EXTRN or ENTRY. If FORTRAN and assembler programs are to be used in the same execution, the main program should be written in FORTRAN. The standard save area linkage conventions should be followed.

3. Most MUSIC system routines (such as TSTIME, NXTPGM, NPRMPT, etc,) may be called from assembler programs. For more information about these routines, consult *Chapter 9. System Subroutines* of this guide.

4. The message `FILE ERROR: CANNOT ADD SPACE TO THIS FILE` may occur for one of the assembler work files SYSUT1, SYSUT2, or SYSUT3 if your assembler program is very large. To correct this problem, supply the following /FILE statement before the /LOAD, using n=1, 2, or 3 as appropriate. The default space allocation is SPACE(200) (i.e. 200K primary space) for SYSUT1, and SPACE(150) for SYSUT2 and SYSUT3.

```
/FILE SYSUTn NAME(&&TEMP) NEW DELETE RECFM(V) SPACE(400)
```

Macro Library - ASM

A macro library containing many OS macro-instructions is automatically available for your use (see below). The user can specify a different macro library by placing a /FILE statement defining the macro library at the beginning of the job. The /FILE statement must have a ddname of SYSLIB and the PDS parameter specified. (Consult the description of the /FILE statement for files.) For example, if the PDS parameter is specified as PDS(*.MAC), then if a macro, say, ABC is used in the program, the file ABC.MAC will be referenced. This means that all you need to do is to create a series of macros stored in files with names

XXX.MAC, where XXX is the name of the macro used. The default /FILE statement for the macro library is /FILE SYSLIB PDS(*.M). If you want to add in any special macros to the default ones, you just create files with names XXX.M as described above. You may also use /INCLUDE statements to bring in any special macros that you may write.

Macro Instructions - ASM

The following is a list of the OS/VS macro instructions provided, showing the extent to which they are supported by the OS/MUSIC interface.

Macro	Comments
ABEND	Completion code and dump options only.
CALL	Non-overlay form only.
CHECK	
CLOSE	LEAVE and REREAD only. REREAD rewinds the data set. CLOSE writes an end-of-file after the last data record.
DCB	All parameters, except SYNAD, supported with the following restrictions: BUFNO 1 only BFTEK S only DSORG PS only MACRF GM,GL,PM,PL,R,W only RECFM F,FA,FM,FB,FBA,FBM only
DCBD	
ESTAE	Specify ABEND exit routine.
FREEMAIN	VC, VU, and R forms only.
GETMAIN	VC, VU, EC, EU, and R only.
GET	
OPEN	INPUT, OUTPUT or INOUT, OUTIN.
POST	
PUT	
READ	SF, length or S only.
RETURN	
SAVE	
SNAP	See the description later on.
SPIE	
STAE	Specify ABEND exit routine.
STIMER	TASK option only. The time interval must be specified as BINTVL or TUINTVL. Timer exit routines are not supported and are ignored if specified.
TGET	
TPUT	
TIME	
TTIMER	TASK option only. The time interval must be specified as BINTVL or TUINTVL. Timer exit routines are not supported and are ignored if specified.
WAIT	
WAITR	
WRITE	SF,length or S only.
WTO	Output on MUSIC unit 6.
WTOR	Input on MUSIC unit 9, output on MUSIC unit 6.

SNAP Macro - ASM

The SNAP macro instruction can be used within assembler programs on MUSIC. This instruction can produce a *snapshot* dump of the program status word (PSW), the general and floating point registers, and specified areas of main storage. The dump can be written to the workstation or to a file. SNAP can also invoke a conversational debug routine, which lets the user inspect and modify main storage, the PSW and the registers before resuming program execution. SNAP is a valuable tool for debugging assembler programs.

For a description of how to use the SNAP macro, refer to the IBM manual *OS/VS2 MVS Supervisor Services and Macro Instructions*, GC28-0683. All of the parameters described can be used on MUSIC, although some (such as TCB, SDATA and STRHDR) are ignored by MUSIC. Since the macro expands differently under MUSIC and MVS, previously written MVS programs using SNAP must be reassembled on MUSIC before they can be run on MUSIC. The execute (MF=E) and list (MF=L) forms are supported.

SNAP is not limited to programs running in OS simulation mode. It can also be used within assembler routines running in MUSIC mode. The macro generates a call to system subroutine SNAPRTN.

The following additional parameters can be used on MUSIC:

UNIT=n
: This parameter can be used instead of the DCB parameter. It specifies the logical unit number (1 to 15) to which the dump is written. For example, UNIT=6 writes the dump to the workstation.

LINE=SHORT (or LINE=S)
LINE=LONG (or LINE=L)
: This specifies whether SNAP is to use short (16 bytes per line) or long (32 bytes per line) dump lines. Short lines are usually more convenient for workstation output. The default is LINE=SHORT for workstation jobs and LINE=LONG for batch jobs.

OPT=C
: This requests that the conversational debug routine be called after the dump, before resuming program execution. The debug routine reads statements from logical unit 9 and writes to logical unit 6. OPT=C is ignored for batch jobs. To change registers or the PSW during debug, use the STORE command to change the values in main storage. The HELP command tells you where the PSW and registers are in main storage.

Notes:

1. When the DCB parameter is used, the SNAP routine will open the file if it is not already open. The data control block (DCB) must specify DSORG=PS, and either MACRF=PM or MACRF=W. The record format must be F, FA, FB, FBA, V, VA, VB, or VBA. Record length may be up to 160. The recommended values are MACRF=PM, RECFM=FBA, LRECL=121.

2. The PDATA parameters JPA, LPA, ALLPA, and SPLS are ignored on MUSIC.

3. The SAH parameter dumps the current save area (pointed to by register 13). The SA parameter dumps the current save area and also gives a save area traceback.

4. All registers are preserved by the SNAP routine (SNAPRTN) except R0, which is set to 0. The expansion of the SNAP macro changes R1. The register values dumped by SNAP are the values before the SNAP macro, except for R1, which has the address of the SNAP argument list.

5. The expansion of the SNAP macro usually contains OI and NI instructions (which change the condition code), so the condition code in the PSW dumped by SNAP is not meaningful.

6. For the execute form of the macro, OPT=C must be specified if wanted, even if it was specified in the

list form. Other options are changed only if specified.

Examples:

```
1.          SNAP  DCB=MYDCB,ID=1,PDATA=(REGS,PSW),OPT=C
                  ...
    MYDCB   DCB   DDNAME=SYSPRINT,DSORG=PS,MACRF=PM,          x
                  RECFM=FBA,LRECL=121,BLKSIZE=1210

2.          SNAP  UNIT=6,ID=2,PDATA=(REGS,PSW,SA),            x
                  STORAGE=(AREA1,END1,AREA2,END2),LINE=LONG
```

Input/Output - ASM

Sequential input/output operations using BSAM and QSAM are supported at execution time. These I/O techniques cannot be used in combination with FORTRAN I/O in a single program.

A standard set of DDNAMES for I/O operations is provided, complete with default values for logical record length.

DDNAME	LRECL	MUSIC I/O
SYSIN	80	input stream (data following /DATA)
SYSPRINT	121	printed output
SYSPUNCH	80	output to holding file (workstation jobs) punched output (batch jobs)
SYSTERM	121	printed output
CONSOLE	80	conversational input (workstation jobs) input stream (batch jobs)

Notes:

1. Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL may be overridden by use of the LRECL= parameter in the DCB macro instruction.

3. For the WTO and WTOR macro instructions, input is performed on MUSIC unit 9 and output on MUSIC unit 6.

4. End-of-file can be indicated on a conversational read by typing /EOF.

Usage - ASM

The Assembler is invoked by the use of a /LOAD ASM statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The Assembler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD ASM specifies that the lines following form a program written for the VS Assembler (and optionally, object modules), and are to be processed by the Assembler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are

ignored.

Parameters: Compile Step - ASM

```
/OPT  {NOLIST}{,NOXREF     }{,NODECK}{,SYSPARM=(xxx)}
      {LIST  }{,XREF(SHORT)}{,DECK  }
              {,XREF(FULL) }
```

NOLIST	Assembled program listing is not to be printed. This is the default for jobs run from a workstation.
LIST	Assembled program listing is to be printed on SYSPRINT. This is the default for jobs run from batch. (If LIST is specified for a job run from a workstation, you can use the PRINT NOGEN or PRINT OFF assembler statements to reduce the amount of output to your workstation.)
NOXREF	No cross-reference list. This is the default for jobs run from a workstation.
XREF(SHORT)	A cross-reference list of all referenced statement labels is to be printed. This is the default for jobs run from batch.
XREF(FULL)	A cross-reference list of all statement labels is to be printed.
NODECK	No object module is to be produced. This is the default.
DECK	An object module is to be written on SYSPUNCH.
SYSPARM	Specifies the value to be assigned to the symbolic variable &SYSPARM.

Notes:

1. Multiple /OPT statements may be used if desired. Other options that can be used are given in the IBM OS/VS Assembler Programmer's Guide publication (GC33-4021).

2. If multiple source programs are in the input stream, they must be separated from each other by /OPT statements. For this purpose, a /OPT statement with no operand may be used.

3. /FILE statements with ddnames of ASM.SYSIN, ASM.SYSPRINT, and ASM.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

Parameters: Loader Step - ASM

```
/JOB  {MAP   }{,NOGO}{,LET}{,NOPRINT}{,DUMP}{,CDUMP}{,DEBUG}
      {FULMAP}

          {,FILRFM}{,NOSEARCH}{,IOTRACE{(C)}}{,SVCTRACE{(C)}}
```

Parameters can be separated by one or more commas or blanks.

MAP	List a storage map of the loaded program.
FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
DUMP	Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.
CDUMP	Request a conversational trace and dump when an error occurs.
DEBUG	Load the DEBUG program into memory for debugging.
FILRFM	Supply the RECFM V if the file is V/VC.
NOSEARCH	Do not search the subroutine library for unresolved routines.
IOTRACE	Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.
SVCTRACE	Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.
2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - ASM

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

Title Lines - ASM

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

References - ASM

IBM System/370 Principles of Operation (GA22-7000)

OS/VS Assembler Language (GC33-4010)

OS/VS Assembler Programmer's Guide (GC33-4021)

OS/VS2 MVS Data Management Services Guide (GC26-3875)

OS/VS2 MVS Data Management Macro Instructions (GC26-3873)

OS/VS2 MVS Supervisor Services and Macro Instructions (GC28-0683)

Examples - ASM

1. An assembler job with options to be used by the assembler and loader. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTER PRT
/LOAD ASM
/JOB MAP
/OPT LIST,XREF

VS Assembler source program 1

/OPT

VS Assembler source program 2

Assembler or COBOL object module(s), if any

/DATA

Data
```

2. A VS Assembler job that reads data from SYSIN (data following /DATA), lists them on SYSPRINT (printed output), and writes them on FILE01 (a temporary UDS file). When end-of-file is encountered on SYSIN, the disk data set FILE01 is rewound, and the card images on this data set are read back and listed.

```
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD ASM
TEST1    START  0
     SAVE   (14,12),,*        SAVE REGISTERS
         BALR   12,0              INITIALIZE BASE REGISTER
         USING  *,12              TELL ASM TO USE IT
         LR     11,13
         LA     13,SAVREG
         ST     11,4(0,13)
         ST     13,8(0,11)
         OPEN   (CARD,(INPUT),PRINT,(OUTPUT))  OPEN SYSIN +
*                                              SYSPRINT
         OPEN   (DISK,(OUTPUT))   OPEN FILE01
LOOP1    GET    CARD,WORK         READ A CARD
         PUT    PRINT,WORK        LIST IT
         PUT    DISK,WORK         WRITE IT TO DISK
         B      LOOP1             DO IT AGAIN
CDEND    CLOSE  (CARD,,DISK,REREAD)    CLOSE SYSIN,CLOSE AND
*                                      REWIND DISK FILE
         WTO    'RECORDS FROM DISK FILE'    SEND MESSAGE
         OPEN   (DISKIN,(INPUT))  RE-OPEN DISK FILE FOR INPUT
LOOP2    GET    DISKIN,WORK       READ RECORD FROM DISK
         PUT    PRINT,WORK        LIST RECORD
         B      LOOP2             DO IT AGAIN
DSKEND   CLOSE  (PRINT,,DISKIN)   CLOSE FILES
         L      13,4(0,13)        RETRIEVE SAVED REG 13
         RETURN (14,12),T         RESTORE REGS AND QUIT
SAVREG   DC     18F'0'            REGISTER SAVE AREA
WORK     DS     80C               WORK AREA
```

```
CARD     DCB    DDNAME=SYSIN,MACRF=(GM),DSORG=PS,                *
                LRECL=80,BLKSIZE=80,RECFM=F,EODAD=CDEND
PRINT    DCB    DDNAME=SYSPRINT,MACRF=(PM),DSORG=PS,             *
                LRECL=80,BLKSIZE=80,RECFM=F
DISK     DCB    DDNAME=FILE01,MACRF=(PM),DSORG=PS,               *
                LRECL=80,BLKSIZE=480,RECFM=FB
DISKIN   DCB    DDNAME=FILE01,MACRF=(GM),DSORG=PS,               *
                LRECL=80,BLKSIZE=480,RECFM=FB,EODAD=DSKEND
         END
/DATA
DATA CARD 1
DATA CARD 2
DATA CARD 3
```

Assembler (Loader) - ASMLG

ASMLG (VS Assembler Load and Go) is the name used to access the loader. The /LOAD ASMLG statement informs MUSIC that the lines following consist exclusively of object modules created by VS Assembler and/or ANS COBOL. (This statement is functionally identical to the /LOAD COBLG statement.) Object modules will load faster if this statement is used instead of /LOAD ASM.

After loading is complete, the program is executed.

Usage - ASMLG

This loader is invoked by the use of the /LOAD ASMLG statement. This statement can be followed by a /OPT SYSIN statement specifying the input unit number and by a /JOB statement specifying parameters for the loading step.

Parameters - ASMLG

```
/JOB  {MAP   }{,NOGO}{,LET}{,NOPRINT}{,NOSEARCH}{,FILRFM}
      {FULMAP}
```

Parameters can be separated by one or more commas or blanks.

MAP	List a storage map of the loaded program.
FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
NOSEARCH	Do not search the subroutine library for unresolved routines.
FILRFM	Supplies a RECFM V if the file is V/VC.

Notes:

1. Multiple /JOB statements may be used if desired.
2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the processors that do not invoke the loader directly.

```
/OPT  {SYSIN=n}
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

Example - ASMLG

```
/LOAD ASMLG
/JOB MAP,LET
/INCLUDE OBJ1     (VS Assembler object module)
/DATA
/INCLUDE DATA1    (user's data)
```

IBM BASIC Environment - IBMBASIC

IBM BASIC (Program Product 5665-948) is an interactive development language. Interactive processing allows you to enter programs and data from a workstation for processing and enables you to verify results. The interactive mode also supports immediate execution of BASIC commands. Your installation may choose not to have this environment available for your use.

Usage - IBMBASIC

Issue the command IBMBASIC to start IBM BASIC. This puts you into the full screen environment. To exit issue the QUIT command.

Notes:

1. IBM BASIC runs in a 3270 type environment in full screen mode.

2. The file IBMBASIC.PROFILE is used at start up time to control the BASIC environment. The user can edit this file and save a modified copy under their own userid. Refer to the *IBM BASIC Programming Guide* for details.

3. The SYSTEM command can be used to issue any MUSIC/SP command. For example

   ```
   CALL SYSTEM 'music command' or
   SYSTEM 'music command'
   ```

   can be used from a BASIC program or in immediate mode.

4. The QUERY command is not supported. The MUSIC/SP LIBRARY command can be used in its place. Use the SYSTEM command to issue the MUSIC/SP LIBRARY command. For example

   ```
   SYSTEM 'LIBRARY *TEST*.BASIC'
   ```

   lists all files that contain TEST in the file name that are BASIC source.

5. Since IBM BASIC programs can invoke facilities of the MVS operating system, some of these are not available on MUSIC/SP. Refer to the *IBM BASIC/MVS Systems Services* manual for more details.

6. The IBM BASIC/GDDM interface is not supported under MUSIC/SP.

7. The IBM BASIC/SQL interface is not supported under MUSIC/SP.

8. File names are limited to 8 characters if not enclosed in quotes. BASIC automatically appends a qualifier to the end of file names not enclosed in quotes. The BASIC qualifiers are:

.BASDATA	data files,
.BASIC	source files,
.BASLOG	log files,
.BASOBJ	workspace files,
.PROFILE	profile files,
.LIST	list files,
.OBJ	object files.

A quoted file name is not automatically qualified.
So, a MUSIC/SP file name can be used
if the file name is enclosed in quotes.
The following two statements are examples of the use of
file names that would be qualified by BASIC.
BASIC would load MYFILE.BASIC and would open MYFILE.BASDATA
respectively.

```
load MYFILE
open #1: 'MYFILE',OUTPUT,NATIVE,SEQUENTIAL,FIXED
```

The following two statements are examples of the use of file names that would be used as is by BASIC. BASIC would load MYFILE and would open MYFILE.OUTPUT respectively.

```
load 'MYFILE'
open #1: "'MYFILE.OUTPUT'",OUTPUT,NATIVE,SEQUENTIAL,FIXED
```

The /FILE definition can be used if the entered file name is the same as a DDNAME that has been setup on a /FILE statement. To define your /FILE definitions to the BASIC environment, make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add your /FILE statements and save it.

9. BASIC automatically creates fixed compressed (FC) files on MUSIC/SP for the following types of files: BASDATA, BASIC, BASOBJ, PROFILE, and OBJ with a record length of 80; BASLOG, LIST with a record length of 133; and for all other FIXED format files with a record length of 80.

   BASIC creates variable compressed (VC) files when you use the VARIABLE option on the BASIC OPEN statement.

   You must create a file outside of BASIC if you need to create a file that is different from the defaults. You can use the following system call to create a file

   ```
   system 'CREATE.BASFILE N(MYTEST) LRECL(250) NEW(REPL)'
   ```

   This call creates the file MYTEST with a record length of 250.

   The program CREATE.BASFILE is the IBM BASIC version of the MUSIC/SP /FILE statement. This allows you to specify any /FILE statement options on the system call to CREATE.BASFILE. After you have defined your file using the system call, you can access this file in your BASIC program via its file name.

   Alternatively, you can use the /FILE statement to define a file to be used within the BASIC environment. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add your /FILE statements and save it.

10. A direct access file must be created initially with dummy records. The program CREATE.DIRECT can be used to create your direct access file. CREATE.DIRECT prompts you for the file name, number of records, and record size. Then it creates the file and fills it with dummy records. Then the direct access file can be used from within your BASIC program. You can use the following system call to create a direct access file from within BASIC

    ```
    system 'CREATE.DIRECT'
    ```

11. The execution of OBJECT files created by IBM BASIC is supported when running within or outside of the BASIC environment. Use the COMPILE command to compile your BASIC program and store the object deck in a file. Use the RUN command to run the object deck within the IBM BASIC environment. For example

```
RUN MYPROG (OBJ
```

runs MYPROG from object module. You can execute the OBJECT file by using the MUSIC/SP linkage editor outside of the IBM BASIC environment.

The performance of using compiled object files within the Interactive BASIC environment is slow. Running from a load module provides better performance.

IBM BASIC programs as load modules

An IBM BASIC program can be run outside of the IBM BASIC environment as a load module. The following example shows the steps to create and run a load module. In this example, the object module for the BASIC program resides in the file MYBASIC.OBJ.

1. Link editing the object module MYBASIC.OBJ to create a load module

```
/FILE LMOD N(MYBASIC.LMOD) NEW(REPL)
/LOAD LKED
/JOB NOGO,MODE=OS
/INC MYBASIC.OBJ
/INC $IBB:IBMBASIC.OBJ
/INC $IBB:MBLIOMRT.OBJ
 ENTRY BLIOMRT
 NAME MYBASIC
```

2. Executing the program MYBASIC

```
/SYS REG=400
/FILE LMOD N(MYBASIC.LMOD)
/LOAD XMON
MYBASIC
```

IBM BASIC Interlanguage Communications

IBM BASIC programs can invoke routines written in Assembler, COBOL, FORTRAN, and PLI. The following example calls the MUSIC/SP subroutine TSUSER. As defined by IBM BASIC interlanguage communications, you must call FLINK with the FORTRAN routine name first before you call the FORTRAN routine.

```
100 INTEGER N
105 CALL FLINK ('TSUSER')
110 CALL FORTRAN ('TSUSER',1,N)
120 PRINT 'The value returned from TSUSER was ',N
130 END
```

There are a few things to keep in mind when you are using VS FORTRAN subroutines with your BASIC program.

1. BASIC character variables have two lengths, the current length and the maximum length. The current length must be assigned to the variable before the variable is used in the calling sequence for the FORTRAN routine. This can be done by assigning spaces to the whole length of the variable. For example

```
DIM A$*8
A$ = '        '
CALL FLINK ('MYFORT')
CALL FORTRAN ('MYFORT',A$)
```

2. If you are doing any I/O within your program and you call a FORTRAN routine, you must define FT05F001 and FT06F001 to IBMBASIC before you start. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add the following two /FILE statements and save it.

```
/FILE FT05F001 RDR
/FILE FT06F001 PRT
```

On MUSIC/SP, when you run a program, a load module is created and executed. To use routines in other languages, you must include these object decks in the link edit that BASIC does behind the scenes. The program that does these steps for you is $IBB:@RUNTIME.LOADLIB. You should make a copy of this file on your userid and add /INC statements for your object modules at link edit time.

Refer to the *IBM BASIC Programming Guide* for further details on interlanguage communications.

IBM BASIC VSAM Support

IBM BASIC supports VSAM under MUSIC/SP. You must first create your VSAM file using the MUSIC/SP utility program AMS. See Chapter 10 of this manual for further details on AMS. In order to use VSAM, you must start BASIC with a /FILE statement for the VSAM file. Make a copy of the file $IBB:IBMBASIC and save it on your userid. Edit it and add a /FILE statement for the VSAM file. Refer to the *IBM BASIC Programming Guide* for further details on how to use VSAM with IBM BASIC.

References - IBMBASIC

B is for BASIC (GS26-4102)

IBM BASIC General Information (GC26-4023)

IBM BASIC Language Reference Summary (SX26-3736)

IBM BASIC Language Reference (GS26-4026)

IBM BASIC Programming Guide (SC26-4027)

IBM BASIC/MVS Systems Services (SC26-4106)

BASIC Compiler - VSBASIC

The VS BASIC compiler is an optional IBM Program Product (5748-XX1) that may be available for your use. VS BASIC programs may be entered and modified using the usual MUSIC techniques or you may use the MUSIC VS BASIC Editor. This editor is designed specifically to handle VS BASIC programs. The writeup for this special editor can be found immediately following this writeup about the VS BASIC Compiler. The standard MUSIC editor may also be used for entering and modifying VS BASIC programs and data.

VS BASIC Highlights

- Arithmetic facilities, in short and long precision, enabling the user to assign values and to perform arithmetic operations on variables and arrays.
- Character facilities, enabling the user to define character variables of different lengths, to concatenate groups of character data items, and to locate and extract substrings within character strings.
- Array facilities, enabling the user to define one and two dimensional character and arithmetic arrays, to re-dimension arrays, to assign a single value to all elements of an array, and to sort arrays into ascending or descending order.
- Stream-oriented file facilities, enabling the user to read and write files, and to reposition files to their beginning or their end.
- Record-oriented input/output functions for both sequential and direct access files.
- Output formatting capabilities, enabling the user to position data items on a print line, and offering commercial formats with comma insertions, asterisk protection, and floating plus, minus, and dollar signs.
- A comprehensive library of built-in mathematical and character functions, performing numerous arithmetic, character-string, and conversion operations.
- User-defined functions, enabling the user to define special-purpose functions not available as built-in functions. Such functions can be defined in one statement or over many statements, and can specify multiple, single, or no arguments.
- Program segmentation statements, enabling the user to sequentially execute a number of separate BASIC programs to fit special needs, and to pass information between these programs.
- Internal constants, supplying commonly-used arithmetic values such as pi and the square root of 2, and providing such metric conversions as inches to centimeters and gallons to liters.

Language Specifications - VSBASIC

The language specifications may be found in the IBM publication *SYSTEM/370 VS BASIC Language* (GC28-8303).

Usage Notes - VSBASIC

- The DELETE FILE, KEY, NOKEY, DUPKEY, NOREC, and DUPREC statements for record-oriented input/output are not supported. This means that they may be coded but will have no effect. REWRITE is only supported for direct files.

- Direct files, those using the REC clause, must be files with a record format of FIXED (F). Note that the system default is FIXED COMPRESSED (FC). FIXED record format files may be created using /FILE statements. For example,

```
/FILE 1 NAME(XXX) NEW LRECL(100) RECFM(F) SPACE(50) NORLSE
/LOAD IEFBR
```

  will create a FIXED format file called XXX with a record length of 100 and space of 50K.

- Record-oriented files must be allocated prior to running the VS BASIC program.

- MUSIC checks the file's record format to determine which type of access is valid. If it is FIXED, the file is assumed to be direct (RRDS), and both sequential and direct access are allowed. For files with any other record formats, or UDS files, the file is assumed to be sequential (ESDS).

- After a FIXED file has been accessed by VS BASIC, the MUSIC /LIST command, and other related commands, may get confused as to the number of lines in the file, since they have not been processed sequentially. Editing the file and saving it back will reset the end of file pointers and cause this problem to disappear.

- The MUSIC subroutine library is not available to VS BASIC programs, and VS BASIC cannot communicate with subroutines written in other languages.

- The EOF clause on the PUT statement is not supported for files (ddnames) SYSIN, SYSPUNCH, SYSPRINT and CONSOLE, unless they are specified on /FILE statements at the beginning of the job.

- When the CHAIN statement is used, the file name is taken to be the name of a file. If the file name of "*" is given then the chained program is assumed to be the next one in sequence in the input stream.

- It is possible to chain to programs written in other languages. To do this, the user must use a slightly modified version of the CHAIN command as follows:

```
CHAIN '%name parameters'
```

  If the first character of the name field is specified as a "%" sign, the program contained in the following file name is executed. Any characters following the file name are passed to the program as parameters.

Examples:

```
250 CHAIN '%FILE1'      execute the program in FILE1
100 CHAIN '%PROG X286' execute the program in PROG.  The string 'X286' is passed as a
                       parameter.
```

- The upward pointing arrow on TTY terminals is treated as if it were the same as the vertical bar (|) character. Therefore use the symbol ** for exponentiation in VS BASIC.

- To conserve main storage, REMARK statements are not passed to the compiler, unless the REM compiler option is specified. Therefore, if REMARK statements are referenced by other statements

such as GOTO, the REM option must be used.

- Source statements are limited to 80 columns in length. See the following discussion of continuation statements for one method of getting around this limitation.

Continuation Statements - VSBASIC

Since VS BASIC source statements on MUSIC are limited to 80 columns, it is occasionally necessary to continue a statement onto another line. This is done by using a line number of the form *n.m* on the continuation lines. *n* is the number of the BASIC statement and m (1 or more) is the continuation line number. A maximum of two continuation lines may be used.

The compiler concatenates the continuation text to the end of the previous line (at column 80). The continuation text is considered to start immediately after the *n.m* line number if the following character is non-blank, or one character after the *n.m* otherwise.

Examples:

```
120 LET X = A + B
120.10 + C + D
120.20 + E + F

200 INPUT A,B,C,
200.1 D,E,F
```

Main Storage Requirements - VSBASIC

The amount of main storage available for the user's program (object code plus array storage) depends upon the amount of buffer space being used and the space required for storing the source statements, but is usually about 25K (K=1024 bytes) in a 108K user region. This limits the size of each VS BASIC program to about 200 to 250 statements. However, several programs may be chained together by use of the CHAIN statement. Alternately, use a "/SYS REGION=nnn" statement to ask for a larger user region. Also, if the VS BASIC compiler and library are kept resident in the system link pack areas (this is done by your MUSIC installation, not by you), then the space available is increased by about 50K.

Control Statement Set Up - VSBASIC

The MUSIC VS BASIC Editor is available to assist in typing in, making corrections and running a BASIC program. This facility is described in more detail in the separate writeup that follows this one. Alternately a VS BASIC program can be entered and run in a method similar to the other MUSIC processors as follows:

```
/FILE statements (if necessary)
/LOAD VSBASIC
/OPT parameters  (see below).
...VS BASIC source program or object module.
...
/DATA
...data
...
```

Notes:

1. On the /LOAD statement, VSBASIC may be abbreviated as VSBAS.

2. Any number of /OPT statements may be used.

3. A parameter specified on a /OPT statement remains in effect until specifically changed by a later /OPT statement.

4. More than one source or object program may be used. They are compiled and executed separately, one after another. Each must be separated from the preceding program by one or more /OPT statements. For this purpose it is permissible to use a /OPT statement which does not contain any parameters. A program may use the CHAIN statement to pass information to the next program.

5. If more than one program is present, only the last one is able to read from the input stream (data following the /DATA statement). However, DATA statements can be used within each source program.

6. Following the /LOAD VSBASIC statement, any control line (other than /OPT, /JOB or /DATA) containing a slash in column 1 is ignored. A /JOB statement is treated exactly like /OPT.

7. The user may type /EOF to indicate end of file on a conversational read.

VS BASIC Object Modules

The VS BASIC compiler can produce an object module representing the BASIC program, and this object module can later be used in place of the source program. However, this module is not a standard object module, and cannot be processed by /LOAD LOADER. For most VS BASIC programs, there is no advantage in using an object module, since the time to compile the source is usually less than the time to load the object module.

VS BASIC Compiler Parameters - /OPT Statement

Parameters on the /OPT control statement are used to specify compiler options, input/output options, and other information. Parameters must be separated by commas and must not contain embedded blanks. One blank must be present between /OPT and the first parameter.

LIST NOLIST	Specifies whether or not a listing of the source program is to be printed. Default is NOLIST for workstations and LIST for batch.
PRINT NOPRINT	NOPRINT suppresses the compile and execution time messages.
GO NOGO	NOGO means that the program is to be compiled but not executed. The default is GO.
SPREC LPREC	SPREC specifies that the program is to use short precision for arithmetic calculations. LPREC specifies long precision. The default is SPREC. SPREC and LPREC are ignored if specified for an object module, since the option in effect at the time the program was compiled is always used.

SYSIN=n This specifies the unit number n from which additional compiler input is to be read. The additional input may contain source, object, and control statements. When end of file or a /DATA statement is reached on the specified unit, reading continues with the lines following the /OPT statement containing the SYSIN=n parameter.

INPUT=n Specifies the input unit number to be used for reads resulting from execution of INPUT statements in the BASIC program. Unit 9 (conversational input) is assumed if this parameter is not used.

REM
NOREM To conserve main storage, REMARK statements are usually not passed to the compiler. Use the REM option if you wish REMARK statements to be processed. If remark statements are referenced by other statements, (such as GOTO), then you must use the REM option. The default is NOREM.

BUF=n Specifies the number of bytes to be reserved for data set buffers. The default is 4096 bytes. If several data sets are used, a value greater than 4096 may result in more efficient input/output. However, specifying a value greater than 4096 decreases the space available for object code and arrays. If the BUF parameter is used, it should appear on the first /OPT statement.

BUFF=n Same as BUF=n.

NOPRINT This parameter causes some information messages to be suppressed, for example the messages giving compile and execution times.

LINESIZE=n Specifies the maximum line length to be used for workstation display during execution of the BASIC program. The default is 120, and the allowable range is 72 to 132.

DECK
NODECK Specifies whether or not an object module is to be produced. The default is NODECK. If DECK is specified, the object module will be written to SYSPUNCH (see the description of ddnames below).

STORE
NOSTORE Same as DECK and NODECK.

Examples of /OPT statements:

```
/OPT LPREC,LIST

/OPT NOPRINT,LINESIZE=100,BUF=8192

/OPT NOGO
```

Control of Input/Output Files - VSBASIC

User files referenced by VS BASIC input/output statements are identified by a file name specified on the BASIC statement. This file name is associated with a MUSIC User Data Set or file in the following way:

1. VS BASIC will first attempt to match the file name with the ddnames defined on /FILE statements at the beginning of the job. A file name referenced this way must be 1 to 8 characters long.

2. If the file name cannot be matched, VS BASIC will try to resolve it by looking for a file with the specified file name in the Save Library. File names referenced this way can be 1 to 17 characters in

length.

3. For stream-oriented files, if the file cannot be found in the Save Library but is to be opened for output, a new file will be created with the specified file name.

Pre-Defined DDNAMES - VSBASIC

ddname	Default LRECL	Maximum LRECL	Minimum LRECL	MUSIC I/O
SYSIN	80	80	20	input stream (data following /DATA)
SYSPRINT	133	133	20	printed output
SYSPUNCH	80	80	20	holding file (workstation jobs) or punched output (batch)
CONSOLE	80	80	20	conversational input (workstation jobs) or input stream (batch)

The pre-defined ddnames can be overridden by specifying /FILE statements with the corresponding ddnames at the beginning of the job.

Examples of VS BASIC Programs

Example 1

```
*Go
list vsbsm2
*In progress
/LOAD VSBASIC
10 REM PROGRAM TO CALCULATE COMPOUND INTEREST
20 PRINT ' '
30 PRINT 'COMPOUND INTEREST PROGRAM...'
40 PRINT ' '
50 PRINT 'ENTER INITIAL AMOUNT'
60 INPUT P
70 PRINT 'ENTER INTEREST RATE (E.G. 4.5)'
80 INPUT I
90 PRINT 'ENTER NO. OF PERIODS PER YEAR'
100 INPUT N
110 PRINT 'ENTER NO. OF YEARS'
120 INPUT Y
130 FOR J=1 TO N*Y
140 P=P+P*I/(100*N)
150 NEXT J
160 PRINT 'FINAL AMOUNT IS',P
170 GO TO 50
180 END
*End
*Go
vsbsm2
*In progress
VS BASIC
COMPILE =   0.12 SEC
```

```
COMPOUND INTEREST PROGRAM...

ENTER INITIAL AMOUNT
?
200.00
ENTER INTEREST RATE (E.G. 4.5)
?
9.5
ENTER NO. OF PERIODS PER YEAR
?
4
ENTER NO. OF YEARS
?
3
FINAL AMOUNT IS   265.0674
ENTER INITIAL AMOUNT
?
/eof
** END OF FILE ON UNIT   9
EXEC =   0.17 SEC
*End
*Go
```

Example 2

```
*Go
list vsbsm3
*In progress
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD VSBASIC
10 REM  THIS PROGRAM READS NUMBERS FROM UNIT 9,
20 REM  WRITES THEM TO THE TEMPORARY UDS FILE01,
30 REM  THEN READS THEM BACK AND PRINTS THEM.
40 INPUT A,B,C
50 IF A=999 GO TO 80
60 PUT 'FILE01',A,B,c
70 GO TO 40
80 CLOSE 'FILE01'
90 GET 'FILE01',A,B,C, EOF 120
100 PRINT A,B,C
110 GO TO 90
120 END
*End
*Go
vsbsm3
*In progress
VS BASIC
COMPILE =   0.12 SEC
?
1.2, 3.4, 5.6
?
10,-20,30
?
999,0,0
     1.2       3.4       5.6
     10        -20       30
EXEC =   0.16 SEC
*End
*Go
```

Example 3

```
*Go
list vsbsm5
*In progress
/LOAD VSBASIC
10 REM THIS PROGRAM READS IN NUMBERS FROM A FILE
20 REM AND DISPLAYS THEM ON THE WORKSTATION.
30 PRINT 'ENTER FILE NAME'
40 INPUT A$
50 GET A$,X,EOF 80
60 PRINT X
70 GOTO 50
80 END
*End
*Go
vsbsm5
*In progress
VS BASIC
COMPILE =   0.12 SEC
ENTER SAVE FILE NAME
dat77
 10
 20
 30
EXEC =   0.13 SEC
*End
*Go
```

Example 4. Copying one sequential file to another.

```
10 DIM A$80
20 READ FILE 'FILE1', A$, EOF 50
30 WRITE FILE 'FILE2', A$
40 GO TO 20
50 PRINT ' END OF COPY'
60 END
```

Example 5. Printing selected records from a direct file.

```
10 DIM A$80
20 PRINT ' ENTER NUMBER OF RECORD TO PRINT'
30 INPUT I
40 IF I=0 GO TO 80
50 READ FILE 'DATAFILE', REC=I, A$
60 PRINT A$
70 GO TO 20
80 END
```

VS BASIC Editor

This VS BASIC Editor provides a convenient method of entering, modifying and running programs written in the VS BASIC Language. It realizes the fact that all BASIC statements have unique line numbers assigned in numerical order.

This editor is a standard feature of the MUSIC system support for the optional IBM VS BASIC compiler.

The normal MUSIC editor (the /EDIT command) may also be used for entering and modifying VS BASIC programs and data files. The major advantages of the VS BASIC Editor are that it is statement number oriented and can renumber BASIC programs.

BASIC statements may be entered in any order, since the editor will automatically arrange them in order of their BASIC line number. Should two or more statements have the same line number, the one entered last will be taken. An entire statement entered previously may be deleted simply by typing in just the BASIC statement number with nothing following it.

Special commands are provided for list, delete, change, save, execute, and renumber operations. These commands refer to lines of the file by their line numbers. Line numbers are of the following form:

nnnnn

or nnnnn.mmm

where *nnnnn* is the 1 to 5-digit BASIC statement number, and *mmm* is an optional 1 to 3-digit continuation statement number. Blanks may optionally be used preceding or following the line number, but the line number itself must not contain embedded blanks.

Types of Input Lines - VSBASIC Editor

The following types of input lines are accepted:

1. Control statements with / in column 1:

   These are accepted only at the beginning of the file. They are automatically assigned line numbers 0.10, 0.20, 0.30, etc. and may be referenced by these line numbers for purposes of replacement, listing, insertion, etc. If no / statements are present, the line '/LOAD VSBASIC' is automatically supplied as the first line of the file.

2. BASIC source statements:

   nnnnn text

   or nnnnn.mmm text

3. Special commands, optionally identified by $ in column 1:

   These commands (described below) enable the user to list, change or delete lines of the file, renumber the BASIC program, turn statement number prompting on or off, save and execute the updated file, and perform various other operations. The commands consist of a command keyword followed by a list of operands. A blank is optional following the command keyword, except for the SAVE, EXEC and FILE commands, where a blank is required. Normally the operands are separated by commas, but one or more blanks (zero or more in the case of CHANGE) may be used instead of a comma.

   A command may be identified as a command by typing a dollar sign ($) before the command name, for example: $CHANGE, $SAVE. The $ character is optional except when line number prompting is active.

Invoking the VS BASIC Editor

The VS BASIC Editor is invoked by the MUSIC command /BASIC, which is used as follows:

```
/BASIC xxxxxx
```

where *xxxxxx* is the name of an existing file which is to be edited. If the name is omitted, then the contents of the existing /INPUT file will be used. When the name NEW or NULFIL is used, the edit begins with an empty file; this is useful when a new program is to be typed in.

The file *xxxxxx* to be edited must contain only MUSIC control statements (at the beginning of the file) and the BASIC source statements for a single BASIC program. The file must not contain a /DATA statement followed by lines of data, but the BASIC DATA statement may be used. Files which do not conform to these restrictions should be edited using the MUSIC Editor.

VS BASIC Editor Commands

```
CHANGE n,/string1/string2/
C
```

This command changes *string1* to *string2* in the specified line n. The new line is displayed. LAST may be specified instead of a line number, meaning the last line of the file. If the line number is omitted, the change is applied to the last line typed in by the user. *string2* may be null (that is, of zero length) but not *string1*. Normally a slash ("/") is used for the string delimiter, but any character other than a digit, letter (A-Z), blank, comma or period may be used instead. The third delimiter may be omitted.

```
DELETE  {n1}{,n2}
DEL
```

This command is similar to LIST, except that the specified line or lines are deleted from the file. A line may also be deleted by simply typing its line number.

```
END
```

This command writes the updated file to unit 10 and terminates the editor. The user must then type a '/SAVE name,SV' or '/SV name' command in order to save the file.

```
EXEC  {name}
EX
```

This command terminates the editor, saves the updated file under the specified name (using a scheduled "/SV name" command), and then executes the BASIC program (using a scheduled "/EXEC name"

command). If no name is specified on the command, then the original file name is used. If there is a possibility that the scheduled /SV command will not be successful, this command should not be used. (If the automatic SAVE does not work, then the changed version of the file will be lost.)

```
FILE  {name}
```

This command saves the updated file, using the specified file name or the name of the original file if no name is specified on the command. The editor is then terminated. If the name NULFIL or NEW was used on the /BASIC command, the user will be prompted to enter a file name.

```
LIST  {n1}{,n2}
L
```

The LIST command displays the specified line or range of lines, or the entire file if no line numbers are specified. LAST may be specified instead of a line number, and means the last line of the file.

```
PRINT  {n1}{,n2}
P
```

The PRINT command is the same as LIST.

```
PROMPT {n,m   }
PR     {OFF   }
       {x     }
       {NOCHAR}
```

The PROMPT command is used either to request statement number prompting or to specify a prompt character.

When n and m are used, the editor begins issuing statement number prompts, using n as the starting number and m as the increment. The increment m is optional, defaulting to the previous m, or to 10 on the first $PROMPT command. If OFF is specified, statement number prompting is terminated. n and m may be from 1 to 99999. Initially prompting is off.

When the PROMPT command is used without any operand, it is taken as "PROMPT n" where *n* is the smallest multiple of the current increment which exceeds the largest line number in the file. This automatically allows lines to be added to the end of the file.

When statement number prompting is in effect, the user may type the text of the statement, a $ command, or a line beginning with an overriding statement number.

When *x* is specified on the command, it is used as a conversational read prompt when statement number prompting is not done. *x* must be a single character other than a letter or digit. The NOCHAR option removes the prompt character. A conversational read prompt is useful for workstations whose keyboards

do not lock.

```
QUIT
```

This command terminates the editor without saving the file.

```
RENUM a,b,c,d
REN
```

This command renumbers the BASIC program. The renumbering process includes scanning for statement number references and altering them appropriately. It is assumed that the program conforms to the language and syntax rules of VS BASIC. It is recommended that the program be compiled (in order to detect and correct language errors) before being renumbered. The parameters are:

a Starting new line number to be used. The default is 10.
b The statement number increment to be used. The default is 10.
c The old statement number of the first line of the range of lines to be renumbered. If this parameter is omitted, renumbering begins at the beginning of the file.
d The old statement number of the last line of the range of lines to be renumbered. If this parameter is omitted, renumbering continues until the end of the file.

```
RUN
RU
```

This command is similar to EXEC, except that the MUSIC input file (/INPUT) is automatically used. Thus the new file replaces the /INPUT file and then the BASIC program is executed from the /INPUT file.

```
SAVE  {name}
SV
```

The SAVE command causes the current updated version of the file being edited to replace the file that has the specified file name, without terminating the editor. This is similar to the FILE command except that the edit session will not be terminated. If no name is specified on the command, then the replacement will be done on the original file being edited. The SAVE command cannot be used to store into the /INPUT file.

Examples of VS BASIC Editor Commands

```
LIST 10,300

LIST 5,20.2

L LAST
```

```
DEL50,50.1

CHANGE 25,/X=1.9/X=0.19/

C25#ABC#DE#

C/LTE/LET/
```

Sample Usage of VS BASIC Editor

The following session illustrates how the VS BASIC Editor may be used to enter a BASIC program, make changes to it, save it, and execute it. Lines typed by the user are shown in lower case; lines displayed at the workstation are shown in upper case.

(Some users find that it is more convenient to use the /INPUT file to temporarily hold a copy of the BASIC program until they have debugged it. This can be done by replacing the /BASIC NUFIL command in the following by /BASIC and by using a RUN command instead of the EXEC FILEPQ given here.)

```
/basic new
*In progress
BASIC
10 rem this program adds 2 numbers
20 print 'enter 2 numbers to be aded
25 input a,b
30 c=a+b
40 go to 25
50 end
35 print 'the sum is',c
c 20 @aded@added'@
20 PRINT 'ENTER 2 NUMBERS TO BE ADDED'
renum
RENUMBERED
list
/LOAD VSBASIC
10 REM THIS PROGRAM ADDS 2 NUMBERS
20 PRINT 'ENTER 2 NUMBERS TO BE ADDED'
30 INPUT A,B
40 C=A+B
50 PRINT 'THE SUM IS',C
60 GO TO 30
70 END
exec filepq
FILED
VS BASIC
COMPILE =   0.12 SEC
ENTER 2 NUMBERS TO BE ADDED
?
1.234, 2.005
THE SUM IS      3.239
?
/eof
**END OF FILE ON UNIT   9
EXEC =   0.06 SEC
*End
*Go
```

IBM C/370 Compilers

Programs written in the C language can be compiled and run using one of two IBM C compilers: IBM C/370 compiler (Program Product 5688-040) or the IBM C/370 Compiler Version 2 (5688-187) and IBM C/370 Compiler Version 2 Library (5688-188).

The C processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

Usage - C/370

The C/370 Compiler is invoked by the use of a /LOAD C370 statement. This processor invokes the C/370 Compiler first, and then the loader, upon successful completion of the compilation process. After loading, the compiled program is automatically executed using the OS/MUSIC interface (unless /JOB NOGO is used or the compiler return code is 12 or more). The /OPT and /JOB control statements may be used with this processor. C/370 object modules can optionally be produced and saved, intermixed with source. The object modules are identical with those produced on OS.

Available Unit Numbers and Buffer Space - C/370

Any tape blocksize up to 32760 may be used, provided the user region size (specified on /SYS REGION=nnn) is large enough.

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the compiler modules.

External File Names - C/370

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

DDNAME	LRECL	MUSIC I/O
SYSIN	80	input stream (data following /DATA)
SYSPRINT	121	printed output
SYSLIN	80	output to holding file (workstation jobs) punched output (batch jobs)
SYSTERM	121	printed output
SUBLIB	-	object modules for C/370 library functions
SYSLIB	80	PDS containing C/370 header files (.H in source)
SYSMSGE	-	PDS containing C/370 compiler messages
SYSCPRT	133	print output (source listing)
SYSUT1	80	compiler work file
SYSUT2	80	compiler work file
SYSUT3	80	compiler work file
SYSUT4	80	compiler work file
SYSUT5	80	compiler work file
SYSUT6	3200	compiler work file
SYSUT7	3200	compiler work file
SYSUT8	3200	compiler work file
SYSUT9	137	compiler work file
SYSUT10	133	compiler work file

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

Note that in some cases, file SYSCPRT must refer to a dataset and not to a workstation or printer. For example, when the LIST parameter is specified, C/370 attempts to read the SYSCPRT file. A workstation or printer cannot be rewound and read, and so the compile step fails.

In order to specify user-defined header files, use the following MUSIC JCL statement:

```
/FILE SYSLIB PDS(*.HDR,$IBC:C#3.*.HDR)
```

End-of-file can be indicated on a conversational read by typing /EOF.

Parameters: Compiler Step - C/370

Compiler options may be specified on a /OPT statement preceding the C/370 source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new external procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The C/370 '#pragma options' statement within the source file may be used in place of a /OPT statement for some options. Refer to the definition of the #pragma statement in the C/370 documentation for a list of those options that may be specified. Options on #pragma statements are not cumulative. Note that options specified on a /OPT statement override #pragma options specified in source.

The default compiler options are listed below. Refer to the *IBM C/370 User's Guide* (SC09-1264-01) for a description of the options and their abbreviations.

Workstation Defaults:	Batch Defaults (if different):
NOAGGREGATE	
NOALIAS	
DECK	
NOEXPMAC	
NOFLAG	
NOGONUM	
LANGLVL(EXTENDED)	
NOLIST	
MARGINS(1,72)	
NOOFFSET	
NOOPTIMIZE	
NOPPONLY	
NORENT	
SEQUENCE(73,80)	
NOSOURCE	SOURCE
TARGET()	
TERMINAL	
NOTEST(SYM,BLOCK,LINE)	
NOUPCONV	
NOXREF	

/FILE statements with ddnames of C370.SYSIN, C370.SYSCPRT, and C370.SYSLIN can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions. To provide more space to any of the default compiler work files SYSUTn ("n" can be 1 through 10; the default work space is 100K) for very large programs, supply the following /FILE statement before /LOAD C370:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(F) LRECL(mmm) SPACE(550) NEW DELETE
```

where *mmm* is the logical record length as given in the table above.

Parameters: Loader Step - C/370

```
/JOB  {MAP    }{,NOGO}{,LET}{,NOPRINT}{,DUMP}{,CDUMP}{,DEBUG}
      {FULMAP}

         {,FILRFM}{,NOSEARCH}{,IOTRACE{(C)}}{,SVCTRACE{(C)}}
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

DUMP Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.

CDUMP Request a conversational trace and dump when an error occurs.

DEBUG Load the DEBUG program into memory for debugging.

FILRFM Supply the RECFM V if the file is V/VC.

NOSEARCH Do not search the subroutine library for unresolved routines.

IOTRACE Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.

SVCTRACE Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. Other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - C/370

Execution-time options can be specified by using a #pragma runopts statement in the source program, or by specifying a "/PARM xxx" statement at the beginning of the job, where *xxx* is the option desired. However, in order for the C/370 runtime routines to recognize this, it is necessary to specify the following statement in your source program:

```
#pragma runopts(execops)
```

since by default no execution-time options are recognized on the /PARM statement.

Regardless of whether execution-time options are specified, arguments to the application program must start with a leading '/'. If this is not present, then no arguments will be passed to the application program. Note that the leading '/' is not passed to the application program.

The /PARM statement should be coded as follows. If no execution-time options are required, the leading '/' is still required if application program arguments are to be supplied to the application program. If there are no application program arguments, the '/' is not required.

```
/PARM execution-time options / application-arguments
```

The default execution-time options are:

HEAP(4K,4K,ANYWHERE,KEEP)
ISAINC(0)
ISASIZE(0)
LANGUAGE(UENGLISH)
NOREPORT
TEST(NONE,*,;)
STAE
SPIE

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB statement for a description of the syntax.)

Title Lines - C/370

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

References - C/370

IBM C/370 User's Guide, (SC09-1264).

Systems Application Architecture Common Programming Interface C Reference - Level 2, (SC09-1308).

IBM C Reference Summary, (SX09-1088).

IBM C/370 Diagnosis Guide and Reference V2, (LY09-1804).

Example - C/370

This famous program writes a message to the workstation.

```
/LOAD C370

  /* This is a sample C/370 Program */

#include <stdio.h>
main ()
{
  printf("Hello, World\n");
}
```

CICS/VM-MUSIC Interface - CICS

MUSIC/SP interfaces directly to CICS/VM (Customer Information Control System/VM) Release 2 (product number 5684-011) running under VM. Translation and preparation of the application source files are performed under MUSIC/SP. When the application is to be tested, an automatic transfer of the required files is performed to one of a pool of CMS accounts, and the application is started. When finished, your MUSIC/SP session resumes.

This facility allows you to develop CICS applications under MUSIC/SP, and then perform your testing as required under CMS without requiring you to own a CMS account.

Usage - CICS

You start the CICS/VM-MUSIC interface by issuing the command "CICS". This brings up the panel displayed in Figure 8.3.

```
------------------------ VM/CICS Interface ------------------ Files:
Command ==>

Create NEW Entry Name:       Type:     (Aapp,Capp,Cdat,Cmap,Papp,Alst)
   Execute Appl. Name:         (Application LIST to be executed)

 _ ACCT00.CAPP
 _ ACCT01.CAPP            Selection Options
 _ ACCT02.CAPP
 _ ACCT03.CAPP              D - Delete entry
 _ ACCT04.CAPP              E - Edit entry
 _ ACCTSET.CMAP             G - Generate Application & Screen MAP
                            L - Create LIST of application & screens
                            T - Translate and compile CICS source
                            V - View entry
                            X - Execute selected application & screens
                                Select the items and then press PF4.


------------------------------------------------------------------------
PFKeys 1-Help 2-Setup 3-Exit 4-Exec 5-Loc 7-Up  8-Dn 10-Refresh 12-Cmd
```

Figure 8.3 - CICS Interface

You can create CICS applications in Cobol, PL/I or Assembler, utilizing the command level Application Programming Interface (API) of CICS. This is commonly known as the EXEC CICS interface. Maps, created with BMS statements are also created with this facility.

Programs can be translated, which replaces the EXEC CICS commands with the appropriate language CALLs and then compiles the program to create an object file. The translator will detect API statements that are not supported and notify the user. Applications containing these statements will not compile.

Basic Mapping Support (BMS) is an interface between CICS and its application programs. BMS commands have a simple generalized form, because formatting information is stored separately in what are called maps.

Each map has two forms, *physical* and *symbolic*.

BMS formats a display by embedding control characters in the data stream. A *physical map* tells it how to do this.

A *symbolic description map* is a source language data structure that the assembler or compiler uses to resolve source program references to fields in the map.

The *command area* allows for the execution of MUSIC commands. In addition, the following commands are handled directly by the VM/CICS interface.

Top	Move to the beginning of the list
Last/Bottom	Move to the bottom of the list
Locate	Find the next occurrence of 'string'

To *create* a new application program file or map, the name of the file and its type must be specified. The name is a 1-8 character name that begins with the letter (A-Z). The remaining characters may be letters or numbers.

The *type* must be one of the following:

AAPP - File contains an Assembler application program
CAPP - File contains a COBOL application program
CDAT - File contains date to be used by the CICS program
PAPP - File contains a PL/I application program
CMAP - File contains BMS macro statements, defining a map
ALST - Name defines an application list that can be executed under the CICS/VM product

To execute an application under the CICS/VM product, the name of an application list must be specified. When selected, the specified application files and maps are accessible to the CICS/VM product running under the VM/CMS environment. When the CICS/VM environment completes control is returned to the CICS Interface.

Application Lists - CICS

CICS applications and Maps can be operated on in a variety of ways from the selection screen. A list of existing applications and maps is presented. To the left of the list is a command option, which may be specified as one of the following:

Options - CICS

D Delete the item from the system. Once deleted, it can NOT be accessed.

E Edit the item. You may edit applications, maps or application lists.

G Create the physical and/or symbolic maps. The file must contain BMS macro statements. The type of map created is specified via the SETUP screen. By default BOTH the physical and symbolic maps are generated.

L Specify which application programs and maps are grouped together to form an application list. The application list can then be specified for execution under the CICS/VM product.

To create a list, place "L" on each item that will be place of the list. You may move between multiple screens. The "L" will be changed to an "*" each time a function key or ENTER key is pressed. Once all the individual items have been selected, the name of the new application list is specified via the "Create NEW Application Name" field. The type must be selected as "alst".

T Translate the EXEC CICS API into source language statements. If the translator completes without any errors, the appropriate compiler/assembler is invoked to create an object file. The file must be a CICS application.

V View the item. You may view applications, maps or application lists.

X Specify which application programs and maps are grouped together and then executed under the CICS/VM product. This option is similar to the "L", but an application list is NOT created. This allows for testing of specific applications before including them in an application list.

Place "X" on each item that will be grouped together for execution. You may move between multiple screens. The "X" will be changed to an "*" each time a function key or ENTER key is pressed. Once all the individual items have been selected, press the F4 key to execute under the CICS/VM product.

Program Function Keys - CICS

F2 Select the VM/CICS Setup facility. Options allow for the specification of translator and compiler options. Refer to the help associated with the SETUP facility for details.

F3 Exit the VM/CICS Interface

F4 Execute the items that have been selected via the X option. You must make the selections prior to pressing this key.

F5 Find the next occurrence of the string specified via the last LOCATE command.

F7 Scroll UP the list of applications programs, lists and maps.

F8 Scroll DOWN the list of application programs, lists and maps.

F10 Refresh the list of application programs, lists and maps, by scanning the library. Normally the list is automatically updated when a new items is created. This automatic updating can be turned-off via a SETUP option.

F12 Retrieve the last command issued from the command area to allow for re-executing or changing the command.

CICS/VM Setup Facility

The Setup screen sets the options used by the VM/CICS interface to translate and compile applications and maps. Make any changes and press ENTER to effect the change.

Changes made to the options can be either on a temporary basis (just for this session) or permanently (retained for future sessions).

Refresh file list — By Default (Y) VM/CICS updates the list of applications and maps after adding a new item. If N is specified, the list will NOT automatically be updated after an add.

Options for BMS macros to produce a DSECT and symbolic map.

Generation Option — Use this option to specify the type of map output. The choices are:

BOTH	- produce the physical map and dsect (default)
MAP	- produce the physical map only
DSECT	- produce the DSECT only

Syslib Search — Specify the search order used by the Assembler to locate the BMS macros used in the created of the Dsect and symbolic maps. The default is $CIC:*.AMAC,*.M. You would normally add additional search lists at the end.

The Source Translator translates the EXEC CICS commands in a CICS/VM program and produces two output files:

- A file containing the translated output, ready for compilation.
- A file containing the translator listing.

Translator Options — Specify the translator options. These are as described in the "CICS/VS Application Programmer's Reference". Note, however, that you must use the abbreviated forms of any options having more than 8 characters. For example, you must specify OS (instead of OPSEQUENCE). Each option is separated by a blank.

```
------------------------ VM/CICS Interface Setup ---------------------

                                  Refresh File List after ADD:   (Y/N)
MAP
   Generation Option:          (Both, Csect, Dsect)
 Syslib Search Order:

Source Translator
  Translator Options:

COBOL
       Compiler type:          (COBOL or COBOL2)
     Compile Options:
 Syslib Search Order:

ASSEMBLER
   Assembler Options:
 Syslib Search Order:

PL/I
     Compile Options:
 Syslib Search Order:
=============================================================================
PF-Keys:  1-Help  3-Exit  5-Temporary Change   12-Exit NO Change
```

Figure 8.4 - CICS Interface Setup

Options for the COBOL Compiler are:

Compiler Type — Specify which Cobol Compiler you are using. COBOL2 (the default), specifies that the VS COBOL II compiler be used. COBOL will utilize the OS/VS COBOL

compiler.

Compiler Options — Specify the COBOL Compiler options to be used when compiling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Compiler to locate the files when the COPY or BASIS verbs are used. The default is $CIC:*.CMAC,*.COPY,*.CUSERMAC. You would normally add additional search lists at the end.

Options for the Assembler are:

Assemble Options — Specify the Assembler options to be used when assembling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Assembler to locate the macros used in the program. The default is $CIC:*.AMAC,*.M,*.AUSERMAC. You would normally add additional search lists at the end.

Options for the PL/I Compiler are:

PL/I Options — Specify the PL/I compiler options to be used when compiling the program. Each option is separated by a comma.

Syslib Search — Specify the search order used by the Compiler to locate the %INCLUDE information used in the program. The default is $CIC:*.PMAC,*.M,*.PUSERMAC. You would normally add additional search lists at the end.

Program Function Keys - CICS

F3 — Save the changes made and return to the main selection screen. The changes are permanently and will be used for all further operations.

F5 — Temporarily save the changes and return to the selection screen. The changes will ONLY be in effect for this session. The next time the VM/CICS interface is started, the permanently saved options will be used.

F12 — Return to the selection screen without changing any options.

References - CICS

CICS/VM Application Programming Guide, (SC33-0570).

CICS/VM General Information, (GC33-0571).

COBOL Compiler - COBOL2

MUSIC supports the OS/VS COBOL II (Program Product 5668-958). The COBOL2 processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

Since COBOL II programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile a COBOL II program which will not execute on MUSIC.

Usage Notes - COBOL2

1. Labeled (standard or nonstandard) magnetic tapes are not recognized as such on MUSIC/SP. If the labels are present on tape, then they will appear as a separate file at the beginning of the tape.

2. The library management and teleprocessing (TCAM) features are not supported.

3. The AIXBLD run-time option is not supported. Dynamic building of VSAM alternate indexes is not available. The alternate index must be created using Access Methods Services (see AMS in *Chapter 10. Utilities*).

4. The MERGE verb is not supported.

5. The SORT control dataset is not used and the use of the SORT-CONTROL special register is not supported and is ignored. The following special sort control registers may be used: SORT-CORE-SIZE, SORT-RETURN, and SORT-MESSAGE. The RERUN clause is not supported.

6. Using QSAM (files whose organization is sequential), the following restrictions apply:

   - Open Extend is not supported. The use of the 'APPEND' on the /FILE statement will provide the same result.
   - File status codes of 34 and 39 are not supported.
   - The I-O option on the OPEN statement is not supported.

7. The following restrictions apply to the COBOL II interactive debugger:

   - COBOL II Debug is supported in line mode only.
   - The ONABEND command is not supported and is ignored.
   - A COBOL II program that calls GDDM functions cannot be debugged.
   - If the COBOL II program being debugged abends due to a program interruption, the COBOL statement in error will be displayed on the workstation and the program and debug environment will be terminated. Control will not return to the debug environment.

Input/Output - COBOL2

Input/output operations using QSAM and VSAM are supported. Thus sequential, keyed, and direct access techniques are available on MUSIC/SP using VSAM, and sequential access is also available using QSAM. These operations require the use of the OS/MUSIC interface. This interface is automatically loaded with the /LOAD COBOL procedure. When object modules are used alone you should use the /LOAD COBLG procedure to load them.

For files whose organization is sequential, this I-O option on the OPEN statement is not supported, and the REWRITE statements cannot be used.

System Names - COBOL2

In COBOL II, the programmer establishes a correspondence between file names defined in the program and external input/output devices by means of a SELECT clause in the FILE-CONTROL section. For example, the COBOL II statement:

```
SELECT CARD-FILE ASSIGN TO UR-2540R-S-SYSIN
```

assigns the file name CARD-FILE to the input stream. The I/O description UR-2540R-S-SYSIN is called the system name.

This system name is made up of four components, the *Device Class* (UR), the *Device Number* (2540R), the *File Organization* (S), and the *external Name* (SYSIN). MUSIC supports the following subset of System Name specifications:

Device Class	UR, UT, or DA
Device Number	2540R, 2540P, 1403, 2400, 2314, or 3330
File Organization	S, AS
External Name	SYSIN, SYSPUNCH, SYSPRINT, SYSTERM, SYSOUT, CONSOLE, SYSDBOUT (or a user-specified name - see Note 1 below)

External Names - COBOL2

A standard set of external names (DDNAMES) is provided, complete with default values for logical record length and blocksize.

EXTERNAL NAME	LRECL	MUSIC I/O
SYSIN	80	input stream (data following /DATA)
SYSPRINT	121	printed output
SYSPUNCH	80	output to holding file (workstation job) punched output (batch job)
SYSTERM	121	printed output
SYSOUT	121	printed output
SYSDBOUT	121	printed output
CONSOLE	80	conversational input (workstation jobs) input stream (batch jobs)

Notes:

1. Additional external names can be defined by specifying /FILE statements with the corresponding external names (ddnames) at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL and BLKSIZE are obtained by use of the RECORD CONTAINS and BLOCK CONTAINS clauses. In other areas, the blocksize and the logical record length are defined by the file description entry.

3. DISPLAY may be used alone or with the UPON CONSOLE or UPON SYSPRINT options. ACCEPT may be used with FROM CONSOLE or FROM SYSIN. For batch jobs, ACCEPT FROM CONSOLE

is equivalent to ACCEPT FROM SYSIN.

4. When a disk output data set is closed, an end-of-file is written at the end of the data set. A COBOL CLOSE followed by an OPEN has the effect of rewinding the file. Data sets except for disk data sets, should always be processed using the LABEL RECORDS ARE OMITTED clause.

5. End-of-file can be indicated on a conversational read by typing /EOF.

6. To override the default work files (SYSUT1 to SYSUT7) for providing more space, define a /FILE statement as the following and place it before /LOAD COBOL:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(V) NEW DELETE SPACE(sss)
```

   where *n* is 1,2,3,4,5,6 or 7 and *sss* is the number of K-bytes wanted. The default is SPACE(100).

7. For unresolved externals you should point to the COBOL 2 subroutine library. Do this by including:

```
/FILE SUBLIB PDS(*COBOL2,*OS)
                                (for load and go)
/FILE SUBLIBOS PDS(*COBOL2,*OS)
                                (for linkage editor)
```

   Those references will be resolved then.

8. The following statement is needed when you use /LOAD LKED to execute a COBOL2 load module:

```
/FILE SUBLIBOS PDS(*COBOL2,*MUS,*OS,*EXT)
```

Usage - COBOL2

The COBOL compiler is invoked by the use of a /LOAD COBOL2 statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The COBOL compiler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD COBOL2 specifies that the lines following are a COBOL II program (and optionally, object modules), and are to be processed by the VS COBOL II Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are ignored. Previously compiled object modules may be included in the input stream.

An input stream consisting entirely of COBOL II object modules (and optionally the VS Assembler object modules) can be loaded more quickly for execution by specifying /LOAD COBLG or /LOAD ASMLG to invoke the OS-mode loader. Alternately the /LOAD LKED can be used to produce a load module from these modules.

Multiple COBOL programs (for example, a main program and subprograms) may appear in a single input stream. Each source program must be separated by a /OPT statement.

For information on the use of VSAM files with COBOL, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface* of this guide.

Sort facilities are provided as described under "Using COBOL's SORT Feature" in *Chapter 10. Utilities* of this guide.

If the COBOL COPY or BASIS verb is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The MUSIC /INCLUDE statement may also be used to perform a function similar to the COBOL COPY verb.

Parameters: Compile Step - COBOL2

```
/OPT  [parms...]   (see below for possible parameters)
```

The specified parameters should be separated by commas, and the last parameter must be followed by a blank. A partial list of parameters and their MUSIC defaults is shown here. Additional parameters may be found in the *IBM COBOL II Programmer's Guide*.

BUF= The amount of main storage allocated to buffers. The default value is BUF=16000, which will allow small and medium size source programs to be compiled relatively quickly. If the message INSUFFICIENT CORE FOR THIS JOB is printed, the user may reduce this value (for example, /OPT BUF=6000).

NOSOURCE Program source statements not to be listed. This is the default for jobs run at the workstation.

SOURCE Program source statements to be printed on SYSPRINT. This is the default for jobs run from batch.

NODECK No object module to be produced. This is the default.

DECK An object module is to be written on SYSPUNCH.

NOSEQ The compiler is not to do sequence checking on the source program statements.

SEQ The compiler is to do sequence checking on the source program statements. This is the default.

Notes:

1. Multiple /OPT statements may be used if desired. For other parameters which may be used on the /OPT statement, refer to the IBM COBOL Programmer's Guide publication.

2. /FILE statements with ddnames of COB.SYSIN, COB.SYSPRINT, and COB.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

Parameters: Loader Step

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP	List a storage map of the loaded program.
FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
DUMP	Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.
CDUMP	Request a conversational trace and dump when an error occurs.
DEBUG	Load the DEBUG program into memory for debugging.
FILRFM	Supply the RECFM V if the file is V/VC.
NOSEARCH	Do not search the subroutine library for unresolved routines.
IOTRACE	Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.
SVCTRACE	Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.
2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - COBOL2

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

Title Lines - COBOL2

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

COBOL II Interactive Debugger

COBTEST, the VS COBOL II debug tool, is a flexible tool for monitoring the execution of the VS COBOL II program. With COBTEST you can suspend program execution, continue execution, skip sections of code, correct errors, display and set variables, set up expected input, and display output.

After filling in the panel fields, the system will:

1. Compile the specified source program.
2. Create an executable program (load module)
3. Start the COBOL program, within the debug environment

In Figure 8.5 the initial panel for COBTEST is displayed.

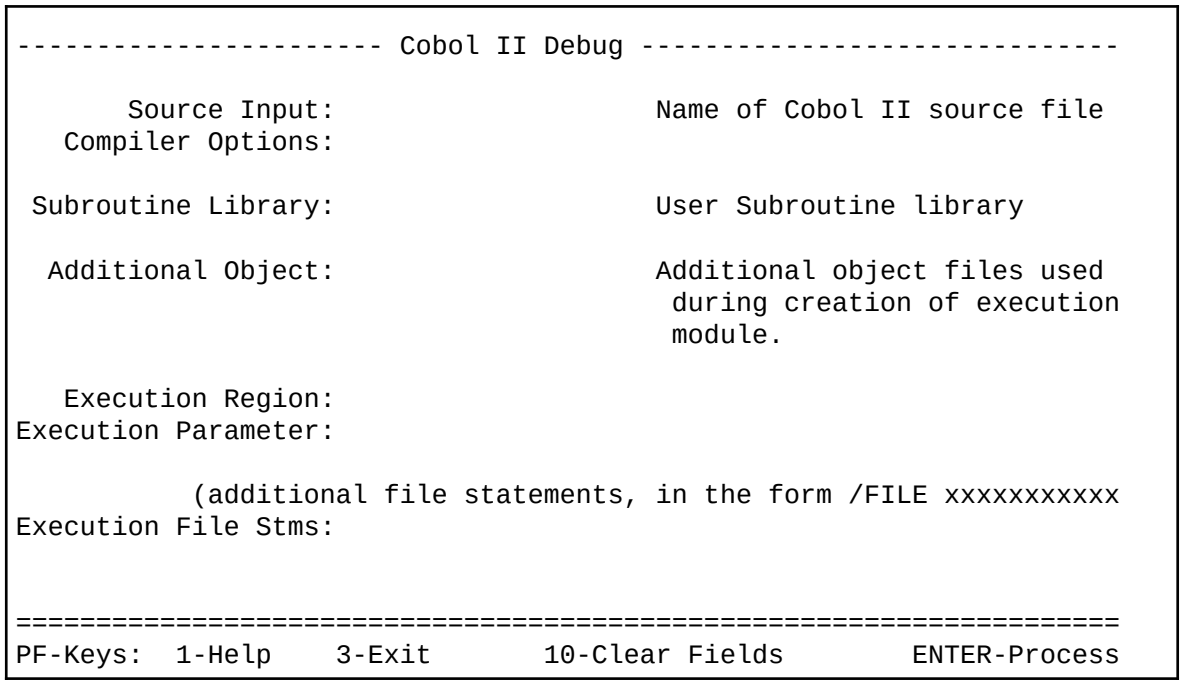

Figure 8.5 - COBOL II Debug

Source Input — Specifies the name of the file containing the COBOL II Source statements that will be compiled and executed under the COBOL II interactive debug environment. This is a required field.

Compiler Options — Specify any COBOL II compiler options that you wish to use during the compile phase. The default options are: Apost, Lib, Source, Deck, Res, Test.

Subroutine Library — Specify the name of **your** subroutine library that you wish to use during the link edit process.

Additional Object — Specify the file names of any other files that will be used during the link edit process. The files **must** be object type files.

Execution Region — Specify the size of the User region to be used during the execution of the program. The default is 800 K.

Execution Parameters — Specify any parameters that your program is expecting at program initiation. You may also specify run-time parameters for the COBOL II environment. The format of this field is: user parameters / run-time parameters.

Execution File Statement — Specify any FILE statements that your program requires during program execution. The complete FILE statement needs to be specified.

By default, information that is written to SYSDBOUT will be placed in the file @@DBOUT. This can be overridden by specifying a FILE statement in the Execution File Statement field.

Keys - COBOL2

3 Terminate the facility. No compilation or program execution will be done.

10 Clear all of the screen fields to the default. All fields are blank, except for execution region which is 800K.

Enter Process the information specified in the different fields and compile, link edit and execute the program under the COBOL II Debug environment.

References - COBOL2

IBM VS COBOL for OS/VS (GC26-3857).

OS/VS COBOL Compiler and Library Programmer's Guide (SC28-6483).

Examples - COBOL2

1. A COBOL job with compiler and loader options. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTOUT PRT
/LOAD COBOL2
/OPT DECK
/JOB MAP
      COBOL Source Program
/OPT
      COBOL Source Subprogram
/DATA
      Data
```

2. A sample COBOL program which scans a file.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    SAMPLE.
       AUTHOR.        IBM PROGRAMMER.
       INSTALLATION.  STL
       DATE-WRITTEN.  MAY 25, 1987.
       DATE-COMPILED.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370.
       OBJECT-COMPUTER.  IBM-370.
       INPUT-OUTPUT SECTION.
         FILE-CONTROL.
            SELECT       INVENTORY
            ASSIGN       INVENT
            FILE STATUS  INVENT-STATUS.
       DATA DIVISION.
       FILE SECTION.
           BLOCK CONTAINS 0 RECORDS
           RECORD 80 CHARACTERS
           LABEL RECORDS STANDARD
           DATA RECORD INVENTORY-RECORD.
       01  INVENTORY-RECORD          PIC X(80).
```

```
   WORKING-STORAGE SECTION.
   77  E-O-F-FLAG                PIC X   VALUE 'N'.
       88  INVENT-E-O-F                  VALUE 'A'.
   77  INVENT-STATUS             PIC XX.
   PROCEDURE DIVISION.
   0000-MAIN-LINE.
        OPEN INPUT INVENTORY
          IF INVENT-STATUS NOT EQUAL ZERO
            THEN DISPLAY 'OPEN ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            GOBACK
          END-IF
        INITIALIZE INVENTORY-RECORD
                   INVENT-STATUS.
   FD  INVENTORY
       RECORDING MODE F
       BLOCK CONTAINS 0 RECORDS
       RECORD 80 CHARACTERS
       LABEL RECORDS STANDARD
       DATA RECORD INVENTORY-RECORD.
   01  INVENTORY-RECORD          PIC X(80).
   WORKING-STORAGE SECTION.
   77  E-O-F-FLAG                PIC X   VALUE 'N'.
       88  INVENT-E-O-F                  VALUE 'A'.
   77  INVENT-STATUS             PIC XX.
   PROCEDURE DIVISION.
   0000-MAIN-LINE.
        OPEN INPUT INVENTORY
          IF INVENT-STATUS NOT EQUAL ZERO
            THEN DISPLAY 'OPEN ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            GOBACK
          END-IF
        INITIALIZE INVENTORY-RECORD
                   INVENT-STATUS.
        PERFORM TEST BEFORE
                UNTIL INVENT-E-O-F
          READ INVENTORY
            AT END
               SET INVENT-E-O-F           TO TRUE
          END-READ
          IF INVENT-STATUS GREATER THAN 10
            THEN DISPLAY 'READ ERROR ON '
-             'FILE INVENTORY' UPON CONSOLE
            SET INVENT-E-O-F             TO TRUE
          END-IF
          DISPLAY INVENTORY-RECORD
        END-PERFORM
        CLOSE INVENTORY
        GOBACK.
```

COBOL Compiler - VSCOBOL

MUSIC supports the OS/VS COBOL (Program Product 5740-CB1). The COBOL processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed with the OS/MUSIC interface.

Notes:

1. Since COBOL programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile an COBOL program which will not execute on MUSIC.

2. MUSIC supports fixed length sequential COBOL files. Programs using indexed, partitioned or direct access techniques may be compiled but not run under MUSIC. Direct access using the relative record technique is supported - see "Direct Access Support" below.

3. For information on the use of VSAM files with COBOL, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface* of this guide.

4. Sort facilities are provided as described under "Using COBOL's SORT Feature" in *Chapter 10. Utilities* of this guide.

5. If the COBOL COPY or BASIS verb is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The MUSIC /INCLUDE statement may also be used to perform a function similar to the COBOL COPY verb.

6. Labeled (standard or nonstandard) magnetic tapes are not recognized as such on MUSIC. If the labels are present on tape, then they will appear as a separate file at the beginning of the tape.

7. The library management, debug and teleprocessing (TCAM) features are not supported.

Input/Output - VSCOBOL

Sequential input/output operations using BSAM and QSAM are supported. These operations require the use of the OS/MUSIC interface. This interface is automatically loaded with the /LOAD COBOL procedure. When object modules are used alone you should use the /LOAD COBLG procedure to load them.

Direct Access Support - VSCOBOL

For COBOL programs to be executed under MUSIC, MUSIC supports the *relative file* organization. This is a direct access technique in which the records of the file are numbered 0,1,2,... This number is called the *nominal key*. Records can be read (by the READ statement) and updated (by the REWRITE statement) randomly by specifying the nominal key. A relative file may also be read (READ statement) and written (WRITE statement) sequentially; in fact, they are normally created using sequential writes.

This requires special care on MUSIC, because sequential access is done using the BSAM access method and random access is done using the BDAM access method, and these two access methods store records differently on disk. BSAM and BDAM storage methods coincide only if the MUSIC record format is F and the

MUSIC record length is a multiple of 512.

For this reason, the /FILE statement defining a COBOL relative record file which is being created sequentially must specify RECFM(F) and LRECL(n), where n is the record length used in the COBOL program, rounded up to a multiple of 512. For example, if the program uses a record length of 100, then the MUSIC file or UDS file must be created as RECFM(F) LRECL(512).

System Names - VSCOBOL

In COBOL, the programmer establishes a correspondence between file names defined in the program and external input/output devices by means of a SELECT clause in the FILE-CONTROL section. For example, the COBOL statement:

```
SELECT CARD-FILE ASSIGN TO UR-2540R-S-SYSIN
```

assigns the file name CARD-FILE to the input stream. The I/O description UR-2540R-S-SYSIN is called the system name.

This system name is made up of four components, the *Device Class* (UR), the *Device Number* (2540R), the *File Organization* (S), and the *external Name* (SYSIN). MUSIC supports the following subset of System Name specifications:

Device Class	UR, UT, or DA
Device Number	2540R, 2540P, 1403, 2400, 2314, or 3330
File Organization	S
External Name	SYSIN, SYSPUNCH, SYSPRINT, SYSTERM, SYSOUT, CONSOLE (or a user-specified name - see Note 1 below)

External Names - VSCOBOL

A standard set of external names (DDNAMES) is provided, complete with default values for logical record length and blocksize.

EXTERNAL NAME	LRECL	MUSIC I/O
SYSIN	80	input stream (data following /DATA)
SYSPRINT	121	printed output
SYSPUNCH	80	output to holding file (workstation job) punched output (batch job)
SYSTERM	121	printed output
SYSOUT	121	printed output
CONSOLE	80	conversational input (workstation jobs) input stream (batch jobs)

Notes:

1. Additional external names can be defined by specifying /FILE statements with the corresponding external names (ddnames) at the beginning of the job. (Consult the description of the /FILE statement.)

2. The default values for LRECL and BLKSIZE are obtained by use of the RECORD CONTAINS and BLOCK CONTAINS clauses. In other areas, the blocksize and the logical record length are defined by

the file description entry.

3. DISPLAY may be used alone or with the UPON CONSOLE or UPON SYSPRINT options. ACCEPT may be used with FROM CONSOLE or FROM SYSIN. For batch jobs, ACCEPT FROM CONSOLE is equivalent to ACCEPT FROM SYSIN.

4. When a disk output data set is closed, an end-of-file is written at the end of the data set. A COBOL CLOSE followed by an OPEN has the effect of rewinding the file. Data sets except for disk data sets, should always be processed using the LABEL RECORDS ARE OMITTED clause.

5. End-of-file can be indicated on a conversational read by typing /EOF.

6. To override the default work files (SYSUT1 to SYSUT5) for providing more space, define a /FILE statement as the following and place it before /LOAD COBOL:

```
/FILE SYSUTn NAME(&&TEMP) RECFM(V) NEW DELETE SPACE(sss)
```

   where *n* is 1,2,3,4, or 5 and *sss* is the number of K-bytes wanted. The default is SPACE(100).

7. The use of the FIPS flagger option in VS COBOL requires a SYSUT6 file. This may be defined by placing a /FILE statement, such as the following, before /LOAD COBOL:

```
/FILE SYSUT6 NAME(&&TEMP) RECFM(V) NEW DELETE
```

Usage - VSCOBOL

The COBOL compiler is invoked by the use of a /LOAD COBOL statement. This statement can be followed by a /OPT statement specifying options for the Assembler and by a /JOB statement specifying parameters for the loading step. The COBOL compiler loads and executes the object code unless /JOB NOGO has been specified.

/LOAD COBOL specifies that the lines following are an COBOL program (and optionally, object modules), and are to be processed by the COBOL Program Product compiler and the resulting object program is to be loaded and executed using the OS/MUSIC interface. /LOAD statements following this statement are ignored. Previously compiled object modules may be included in the input stream.

An input stream consisting entirely of COBOL object modules (and optionally the VS Assembler object modules) can be loaded more quickly for execution by specifying /LOAD COBLG or /LOAD ASMLG to invoke the OS-mode loader. Alternately the /LOAD LKED can be used to produce a load module from these modules.

Multiple COBOL programs (for example, a main program and subprograms) may appear in a single input stream. Each source program must be separated by a /OPT statement.

Parameters: Compile Step - VSCOBOL

```
/OPT  [parms...]   (see below for possible parameters)
```

The specified parameters should be separated by commas, and the last parameter must be followed by a

blank. A partial list of parameters and their MUSIC defaults is shown here. Additional parameters may be found in the *IBM COBOL Programmer's Guide*. The following parameters cannot be used: SYMDMP, RESIDENT, DYNAM, ENDJOB, TEST.

BUF= The amount of main storage allocated to buffers. The default value is BUF=16000, which will allow small and medium size source programs to be compiled relatively quickly. If the message INSUFFICIENT CORE FOR THIS JOB is printed, the user may reduce this value (for example, /OPT BUF=6000).

NOSOURCE Program source statements not to be listed. This is the default for jobs run at the workstation.

SOURCE Program source statements to be printed on SYSPRINT. This is the default for jobs run from batch.

NODECK No object module to be produced. This is the default.

DECK An object module is to be written on SYSPUNCH.

NOSEQ The compiler is not to do sequence checking on the source program statements. This is the default.

SEQ The compiler is to do sequence checking on the source program statements.

Notes:

1. Multiple /OPT statements may be used if desired. For other parameters which may be used on the /OPT statement, refer to the IBM COBOL Programmer's Guide publication.

2. /FILE statements with ddnames of COB.SYSIN, COB.SYSPRINT, and COB.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

Parameters: Loader Step

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing

subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

DUMP Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.

CDUMP Request a conversational trace and dump when an error occurs.

DEBUG Load the DEBUG program into memory for debugging.

FILRFM Supply the RECFM V if the file is V/VC.

NOSEARCH Do not search the subroutine library for unresolved routines.

IOTRACE Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.

SVCTRACE Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - VSCOBOL

Parameters can be passed to the GO step by specifying a "/PARM xxx" statement at the beginning of the job. *xxx* is the parameter desired.

SVC instructions and I/O operations can be traced during execution of the program. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the description of /JOB statement above.)

Title Lines - VSCOBOL

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

References - VSCOBOL

IBM VS COBOL for OS/VS (GC26-3857)

OS/VS COBOL Programmer's Guide (SC28-6483)

Examples - VSCOBOL

1. An COBOL job with compiler and loader options. /FILE statements for GO step (execution time) ddnames are also defined.

```
/FILE GO.CARDIN RDR
/FILE GO.PRINTOUT PRT
/LOAD COBOL
/OPT DECK
/JOB MAP
      COBOL Source Program
/OPT
      COBOL Source Subprogram
/DATA
      Data
```

2. An COBOL program which reads data from SYSIN (data following /DATA), writes them on SYSPRINT (printed output), SYSPUNCH (holding file), and FILE01 (a temporary UDS file). At end-of-file on SYSIN, additional data is read conversationally from CONSOLE (workstation input) until the user signals end-of-file on the workstation by typing /EOF. Then the disk data set FILE01 is read back in and listed.

```
/FILE FILE01 UDS(&&TEMP) NREC(100)
/LOAD COBOL
       ID DIVISION.
       PROGRAM-ID. COBOL-IO-TEST.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CARD-FILE ASSIGN TO UR-S-SYSIN.
           SELECT DISK-FILE ASSIGN TO DA-S-FILE01.
           SELECT CON-FILE ASSIGN TO UT-S-CONSOLE.
           SELECT PRINT-FILE ASSIGN TO UR-S-SYSPRINT.
           SELECT PUNCH-FILE ASSIGN TO UR-S-SYSPUNCH.
       DATA DIVISION.
       FILE SECTION.
       FD  DISK-FILE
           LABEL RECORDS ARE STANDARD
           RECORDING MODE IS F.
       01  DISK-REC  PIC X(80).
       FD  CON-FILE
           LABEL RECORDS ARE OMITTED.
       01  CON-REC  PIC  X(80).
       FD  CARD-FILE
           LABEL RECORDS ARE OMITTED.
       01  CARD-REC  PIC X(80).
       FD  PRINT-FILE
```

```
           LABEL RECORDS ARE OMITTED.
       01  PRINT-REC.
          05  CC    PIC  X.
          05  PRINT-LINE   PIC  X(132).
       FD  PUNCH-FILE
           LABEL RECORDS ARE OMITTED.
       01  PUNCH-REC  PIC  X(80).
       WORKING-STORAGE SECTION.
       77  HOLD1  PIC  9(4)  VALUE ZERO  COMP-3.
       PROCEDURE DIVISION.
           OPEN INPUT CARD-FILE OUTPUT PRINT-FILE, PUNCH-FILE
           DISK-FILE.
       START-HERE.
           READ CARD-FILE AT END GO TO EOF.
           MOVE CARD-REC TO PRINT-LINE.
           WRITE PRINT-REC AFTER POSITIONING 2 LINES.
           WRITE PUNCH-REC FROM CARD-REC.
           WRITE DISK-REC FROM CARD-REC.
           GO TO START-HERE.
       EOF.  DISPLAY ' THIS IS THE END NOW TRY INPUT FROM CONSOLE'.
           CLOSE CARD-FILE PRINT-FILE PUNCH-FILE.
             OPEN INPUT CON-FILE.
       CON-SECT.
           READ CON-FILE AT END GO TO VERY-END.
           DISPLAY 'FROM CONSOLE ' CON-REC.
           GO TO CON-SECT.
       VERY-END.
           CLOSE CON-FILE DISK-FILE. DISPLAY 'CONV END'.
           OPEN INPUT DISK-FILE.
           DISPLAY ' THE FOLLOWING RECORDS ARE FROM DISK FILE'.
       DISK-READ.
           READ DISK-FILE AT END GO TO FINAL-EXIT.
           DISPLAY ' RECORD FROM DISK ' DISK-REC.
           GO TO DISK-READ.
       FINAL-EXIT.
           CLOSE DISK-FILE. DISPLAY ' THATS ALL FOLKS..'.
           STOP RUN.
/DATA
FIRST DATA CARD
SECOND DATA CARD
THIRD DATA CARD
LAST DATA CARD
```

COBOL (Loader) - COBLG

COBLG (ANS COBOL Load and Go) is the name used to access the loader. The /LOAD COBLG statement specifies that the lines following consist exclusively of object modules created by ANS COBOL and/or VS Assembler. (This statement is functionally identical to the /LOAD ASMLG statement.)

Usage - COBOLG

This loader is invoked by the use of the /LOAD COBLG or /LOAD ASMLG statement. This statement can be followed by a /OPT SYSIN statement specifying the input unit number and by a /JOB statement specifying parameters for the loading step.

Parameters

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,NOSEARCH][,FILRFM]
      [FULMAP]
```

Parameters can be separated by one or more commas or blanks.

MAP	List a storage map of the loaded program.
FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
NOSEARCH	Do not search the subroutine library for unresolved routines.
FILRFM	Supplies a RECFM V if the file is V/VC.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters for OS-mode processors are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

```
/OPT  [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

FORTRAN Compiler - VSFORT

Programs written in the Fortran language can be compiled and run using the IBM VS Fortran Compiler and Library. VS Fortran supports both the Fortran 77 and Fortran 66 language standards. Fortran 77, the newer standard, is the default. To request Fortran 66, you must specify the compiler option LANGLVL(66).

MUSIC has a tutorial for learning VS Fortran. Type "TUT" in *Go mode or select the "Tutorials" item for the FSI main menu.

Available Versions - VSFORT

Each MUSIC/SP system provides either of two versions of VS Fortran (but not both): (1) VS Fortran Version 1 (Release 4) Compiler and Library, program number 5748-FO3, or (2) VS Fortran Version 2 Compiler, Library and Interactive Debug, program number 5668-806. Which version is present is an installation option and varies from site to site. Both versions are invoked by the /LOAD VSFORT control statement.

To see which version is available on your system, run the following job and examine the first digit of the level number in the heading line of the output:

```
/LOAD VSFORT
/JOB NOEDIT
      STOP
      END
```

Version 2 is compatible with Version 1, in that it can run all object modules and load modules produced by Version 1. Version 2 accepts Fortran source prepared for Version 1, but the reverse is not true.

VS Fortran Version 2 provides the following enhancements: 31-character variable names, language keywords in mixed upper and lower case (it is not case sensitive), an Intercompilation Analyzer (ICA), graphic relational operators (combinations of "<", ">" and "="), a source level Interactive Debug Facility (IAD), and other features not present in Version 1.

Interactive Debug - VSFORT

With VS Fortran Version 2, MUSIC users can debug and test their programs at the source language level using either IBM's Interactive Debug (IAD) in line mode or MUSIC's full-screen VS Fortran Interactive Debug (TESTF). TESTF is described in the next section of this chapter. IAD is described later in this section. With VS Fortran Version 1, TESTF is available but IAD is not.

Control Statements - VSFORT

VS Fortran in invoked by the /LOAD VSFORT statement. The VSFORT processor accepts source and/or object modules as input, compiles the source, invokes the OS-mode Loader to load the object modules and required library routines into memory, and executes the program. Both the compile and execution steps run in MVS (OS) simulation mode.

The /LOAD VSFORT statement can be preceded by a /SYS statement specifying job region size, by a /PARM statement specifying execution-time parameters (which may include the option FDEBUG to invoke Interactive Debug - IAD), and by /FILE statements defining files for the program. The /LOAD VSFORT statement can be followed by a /OPT statement specifying compiler options, and by a /JOB statement specifying Loader and MVS simulator options.

Control statements are used as follows:

```
/SYS REGION=n                <--- job region size in K
/PARM xxxxx                  <--- run-time parameters
/FILE nn NAME(filename) ...  <--- 0 or more /FILE statements
/FILE ddname NAME(filename) ...
/LOAD VSFORT
/JOB option,option,...       <--- Loader and MVS simulator options
/OPT option,option,...       <--- compiler options
...source program and/or object modules...
/DATA
...data records (Fortran unit 5)...
```

Fortran source, object modules, and data records can be specified by /INCLUDE statements pointing to other files, or the source or data can be contained in the same file as /LOAD VSFORT.

Most compiles require a region size of at least 512K (/SYS REGION=512). Large programs may require 1024K or more.

Defining Program Files - VSFORT

A file is connected to a program by a unit number (also called a data set reference number), which is a number from 1 to 99, or by a ddname (data definition name), which is a 1 to 8-character name. The Fortran convention for ddnames is FTnnF001, where nn is a 2-digit number from 01 to 99. Fortran source statements such as READ and WRITE refer to unit numbers. A Fortran OPEN statement can be used to associate a unit number with a ddname. If no OPEN statement is used, unit number nn is automatically associated with ddname FTnnF001.

Unit numbers and ddnames are defined on /FILE statements which precede the /LOAD statement:

```
/FILE nn NAME(filename) options          <--- unit number

/FILE ddname NAME(filename) options      <--- ddname
```

On a /FILE, ddname FTnnF001 is equivalent to unit number nn. Think of unit number nn on a /FILE as shorthand for ddname FTnnF001.

The following unit numbers and ddnames are automatically defined for VS Fortran jobs:

Ddname	Unit	Record Length	Definition
FT05F001	5	80	Input data following /DATA
FT06F001	6	133	Output to the workstation (for batch jobs: output to printer)
FT09F001	9	80	Conversional input from the workstation (for batch jobs: same as unit 5)

End of data can be indicated on conversational input from the workstation by typing /eof.

Compiler Options on the /OPT Statement - VSFORT

Options for the VS Fortran compiler are specified on a /OPT statement. The options are separated by commas. If necessary, more than one /OPT can be used; their effect is cumulative.

The available options are described in the appropriate VS Fortran Programming Guide publication. MUSIC uses the standard IBM defaults. Some common compiler options are:

NOSOURCE
SOURCE — Produces a source listing on ddname SYSPRINT (the workstation by default). Default is NOSOURCE for workstation jobs and SOURCE for batch jobs.

NODECK
DECK — Produces an object module on ddname SYSPUNCH. The default is NODECK.

FLAG(I)
FLAG(W)
FLAG(E)
FLAG(S) — Specifies which level of compiler messages will be produced: all (I), warnings and above (W), errors and above (E), or only severe errors (S). The default is FLAG(I).

OPT(n) — Optimization level, 0 to 3. OPT(0) is no optimization, which gives faster compiles but produces less efficient code. OPT(1) is recommended for most production programs. The default is OPT(0).

NOSDUMP
SDUMP — Incorporates information on source variable names and statement numbers into the object module. This makes the program bigger but enables symbolic dumps and debugging (IAD) at execution time. The default is SDUMP.

CHARLEN(n) — Maximum length of CHARACTER variables. The default is CHARLEN(500).

LANGLVL(66)
LANGLVL(77) — The Fortran language level to be accepted. The default is LANGLVL(77). If necessary, specify LANGLVL(66) for compatibility with Fortran G1 source.

NOMAP
MAP — Produces a table of variable names and statement labels. The default is NOMAP.

NOXREF
XREF — Produces a table of variable names and statement labels, plus the statement numbers where each is referenced. The default is NOXREF.

NOLIST
LIST — Shows the assembler instructions generated by the compiler. This is useful if you wish to use the assembler-level Debug Facility to debug your program. The default is NOLIST.

Loader and MVS Simulator Options on the /JOB Statement

Options for the MUSIC Loader and for the execution-time MVS simulator are specified on a /JOB statement. The options are separated by commas or blanks. If necessary, more than one /JOB can be used; their effect is cumulative. When using /LOAD XMON to execute a load module, the same options may be used on the program name statement, after the load module ddname or N(filename) specification.

The available /JOB options are:

MAP	Produces a Loader storage map.
FULMAP	Produces an extended Loader storage map.
NOGO	Compile only. Do no load or execute the program.
LET	Allows the program to execute even if unresolved external references (such as missing subroutines) are found during the Loader step.
NOSEARCH	Tells the Loader not to search the Subroutine Library for unresolved external references occurring in the program. With this option, the Loader does not load any subroutines from the Subroutine Library.
NOPRINT	Omit Loader information messages.
SVCTRACE	Traces SVC (supervisor call) instructions during program execution. Abbreviation: SVCTR
IOTRACE	Traces input/output operations during program execution. Abbreviation: IOTR
DUMP	Produces a storage dump if the program ends abnormally or is terminated by an error detected by the MVS simulator. Abbreviation: DU
CDUMP	Allow interactive display of storage if the program ends abnormally or is terminated by an error detected by the MVS simulator. If used in combination with SVCTRACE or IOTRACE, CDUMP stops the program after each SVC or I/O and allows you to display and modify storage interactively. Abbreviation: CDU.
DEBUG	Invokes the MUSIC assembler-level Debug Facility at the start of program execution.
NONRENT	Forces re-entrant modules in the Link Pack Area (LPA) to be loaded into the user region. This is required if you wish to use the assembler-level Debug Facility to trace such modules or set break points in them.
NOEDIT	Suppresses editing of compiler output to the workstation. Normally MUSIC omits or shortens some headings and other information messages during compile. The NOEDIT option allows the full output to appear.
V(n)	This option, where n is a record length value, causes the VS Fortran Library routines to see variable-length MUSIC files (record format V or VC) as fixed-length MVS files (record format F or FB), with a record length equal to the larger of n and f, where f is the MUSIC record length of the file. This option applies only to ddnames FTnnF001. For example, /JOB V(300) causes VS Fortran to treat a RECFM(VC) file with MUSIC record length 200 as a fixed format file with record length 300. This allows new output records to be up to 300 bytes long and avoids the 8 bytes of control information (MVS record and block descriptor words) at the start of each record. The option does not affect the actual record format of the resulting MUSIC file. Note that a new V or VC file has a MUSIC record length of 0. V(256) is automatically assumed for FT06F001.
FILRFM	This option causes MUSIC record format V or VC files to be treated as variable length (MVS record format VB) files by the VS Fortran Library routines. This option may be required in order to produce the MVS record and block descriptor words at the start of each logical record.

Execution-Time Parameters - VSFORT

Fortran run-time options, if required, are specified on a /PARM statement preceding the /LOAD VSFORT or /LOAD XMON statement. All run-time options listed in the VS Fortran Programming Guide can be specified, except that the DEBUG option must be entered as FDEBUG on MUSIC. Thus, the statement for invoking Interactive Debug (IAD) at run time is /PARM FDEBUG (not /PARM DEBUG).

Any options or parameters on /PARM are also accessible to the program via a call to the system subroutines PARM or PARMLC. PARMLC is the same as PARM, except that the parameter string is not converted to upper case.

Block Letter Titles - VSFORT

For a compile in a batch job, one or two lines of block letter titles can be printed preceding the source listing. Each title line is from 1 to 10 characters. The titles are printed twice on consecutive pages, so that one will always be face up after page tear off. The titles are ignored if the job is run at a workstation. The titles are specified by control statements following the /LOAD statement:

```
=TITLE xxxxxxxxxx             <--- first block letter title

=TITLE2 yyyyyyyyyy            <--- second block letter title
```

For example:

```
/LOAD VSFORT
/OPT SOURCE
=TITLE PROG5 VER1
/INCLUDE PROG5.S         <--- program source file
```

Using Object Modules and Load Modules - VSFORT

To create an object module in file prog.obj, use the compiler DECK option as in this example:

```
/SYS REGION=1024
/FILE SYSPUNCH NAME(prog.obj) NEW    <--- object output
/LOAD VSFORT
/JOB NOGO
/OPT DECK,OPT(1)        <--- other options as desired
/INCLUDE prog.s         <--- program source file
```

The MUSIC convention is that source module file names end in ".S", object module file names end in ".OBJ", and load module file names end in ".LMOD". You may use a different convention if you wish.

Once you have object modules for all the routines making up your program, you can create a load module in file prog.lmod as in this example:

```
/FILE LMOD NAME(prog.lmod) NEW SPACE(200)
/LOAD LKED
/JOB MAP,NOGO,MODE=OS,NAME=progname
.ORG 4A00
/INCLUDE prog.obj
```

```
/INCLUDE sub1.obj
/INCLUDE sub2.OBJ
```

To execute the program using the load module, use the following control statements:

```
/SYS REGION=n                <--- desired region size
/FILE ...                    <--- /FILE statements as needed
/LOAD XMON
progname N(prog.lmod) option,option,...  <--- same opts as /JOB
...data records if any...
```

Using a load module to execute your program gives much faster job start up.

Usage Notes - VSFORT

1. Since VS Fortran programs can invoke features of the MVS operating system which are not supported by MUSIC's MVS simulator, it is possible to compile Fortran programs which do not execute successfully on MUSIC.

2. MUSIC supports use of VSAM files with VS Fortran. Refer to the section on VSAM elsewhere in this manual.

3. Features of VS Fortran that depend on MVS dynamic file allocation are not supported on MUSIC. However, the MUSIC subroutines OPNFIL and CLSFIL can be used to access MUSIC files dynamically (without using a /FILE statement). Refer to the chapter on System Subroutines in this manual.

4. Files for VS Fortran direct access must be created as record format F (fixed length, uncompressed). Each direct access record starts on a new 512-byte block on disk, and occupies one or more blocks. This storage method is inefficient in terms of disk space when the record length is small (256 bytes or less). The feature of VS Fortran that automatically formats a new direct access file is not available on MUSIC. The OPEN statement must not specify STATUS='NEW'; omit the STATUS keyword or specify STATUS='OLD'. A new direct access file must be initialized using the ZERO.FILE utility (or equivalent) before being used in a VS Fortran program, unless the program writes all the records in order (1,2,3,...) to the end of the file. Once the file has been initialized, it can be accessed randomly. This example shows creating and initializing a direct access file:

   ```
   /FILE 1 NAME(DA.SAMP) NEW RECFM(F) LRECL(400) SPACE(100) NORLSE
   /INC ZERO.FILE
   ```

5. The VS Fortran INCLUDE statement can be used to incorporate source from an external file into the program. The following format of INCLUDE must be used:

   ```
   INCLUDE (member) n
   ```

   "member" is a 1 to 8-character member name. "n" is an optional value which the compiler uses to decide whether or not to include the source. By default, the contents of MUSIC file member.VSFORT is included into the program. To use a different naming convention for included files, provide a /FILE SYSLIB containing a PDS parameter. The default SYSLIB is /FILE SYSLIB PDS(*.VSFORT). INCLUDE statements must not be nested; that is, the included file may not contain another INCLUDE statement. The MUSIC /INCLUDE statement (which allows nesting) can be used as an alternative to the VS Fortran INCLUDE statement.

Interlanguage Communication - VSFORT

It is sometimes desirable to invoke subprograms written in other programming languages. A VS FORTRAN main program may call subroutines written in Assembler, PL/I, VS/PASCAL, COBOL, and VS FORTRAN.

You may also invoke VS FORTRAN subroutines from Assembler, COBOL, PL/I, and VS/PASCAL. When calling VS FORTRAN subroutines from other languages (where the main program is NOT VS FORTRAN), you must call a subroutine VSFINT before any calls to VS FORTRAN routines. This routine (VSFINT) performs library initialization for the VS FORTRAN environment.

The format of the VSFINT call is:

VS/PASCAL

```
 var
   I            :  INTEGER;
  PROCEDURE VSFINT(VAR I : INTEGER);
   FORTRAN;
begin
   VSFINT(I);
     .
     .
```

VS/COBOL

```
ID DIVISION.
PROGRAM-ID. COBOL-TEST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77     AREAL  USAGE IS COMPUTATIONAL-2.
PROCEDURE DIVISION.
START-HERE.
    CALL 'VSFINT'  USING AREAL.
         .
         .
```

PL/I

```
TOSQ: PROC OPTIONS(MAIN);
      DCL SQUARE ENTRY EXTERNAL;
      DCL VSFINT ENTRY EXTERNAL;
      IREAL=4;
      CALL VSFINT(IREAL);
           .
           .
```

There is an argument passed in the call to VSFINT. This argument is NOT used by the subroutine, and can be of any value.

MUSIC Extensions to NAMELIST Input

MUSIC allows a more convenient form of Fortran NAMELIST input data if the object module MUSNL.OBJ is included with the program. NAMELIST output is not affected. If MUSNL.OBJ is not present, the standard form of NAMELIST input, as described in the language manual, is used.

When /INCLUDE MUSNL.OBJ is placed after the Fortran program, or in the Linkage Editor input when creating a load module, the NAMELIST start (&name) and end (&END) indicators are not used in input data records. The first data item can start at or after column 1 and can extend to column 80. To continue a record, end it with a comma. For a logical variable, for example SWITCH, specifying "SWITCH" is equivalent to "SWITCH=.TRUE." and "NOSWITCH" is equivalent to "SWITCH=.FALSE." When an character string is specified for an array, the remaining elements of the array are set to blanks.

Subscripted array items, floating point exponents, REAL*16 and complex values may not be used. A method of assigning values to particular items of an array is, for example, A=10,3*,20, which sets A(1)=10 and A(5)=20 but does not change A(2) through A(4). The ERRSET routine cannot be used to get control of conversion or other errors. For VS Fortran Version 2, variable names may be up to 31 characters long.

The sample program given below shows an example of MUSIC NAMELIST input.

VS Fortran Interactive Debug (IAD)

The IBM version of Interactive Debug (IAD) is invoked by specifying the run-time option FDEBUG. This is done by placing the statement /PARM FDEBUG before the /LOAD VSFORT or /LOAD XMON statement.

IAD is provided only with VS Fortran Version 2. Since ISPF is not supported on MUSIC, only the line and batch modes of IAD are available. Full screen mode is not available. If full screen mode debugging is needed or you wish to debug with VS Fortran Version 1, use the MUSIC version of VS Fortran Interactive Debug (TESTF).

The only requirement for using IAD is that the routines to be debugged must be compiled with the SDUMP option (which is the default). Also, the compiler option OPT(0) is recommended, to suppress optimization. Some extra ddnames are required for IAD, and the job region size must be increased by at least 500K. To simplify the control statements, file FDEBUG is provided, which contains /PARM FDEBUG, specifies a region size of 1024K, and defines the AFF-- ddnames required for IAD. In most cases, it is sufficient to place the statement /INCLUDE FDEBUG before the /LOAD.

When IAD is run on batch, the AFFIN file must not be empty. Do this by providing an overriding /FILE AFFIN pointing to the file containing the IAD commands.

The IAD HELP command invokes the MUSIC Help Facility to display full screen help information. The IAD SYSCMD command (abbreviation SYS) can be used to enter a MUSIC command while debugging. For example, "sys edit prog.listing". SYSCMD cannot be used within an IAD command list. To signal an attention interrupt to IAD, press the BREAK key (PA1 on a 3270-type workstation) and enter a blank line while in MUSIC attention mode. For the IAD TERMIO command to take effect, the desired unit numbers must be specified by the DEBUNIT run-time option. For example:

```
/PARM FDEBUG,DEBUNIT(5,6,9)
```

The following is a sample Interactive Debug session.

```
/list fdebug.sample
*In progress
```

```
/INCLUDE FDEBUG
/LOAD VSFORT
      READ(5,*) DIAM
      CALL CALC(DIAM,CIRCUM,AREA)
      WRITE(6,10) DIAM,CIRCUM,AREA
10    FORMAT(' DIAMETER=',F8.3,'  CIRCUMFERENCE=',F10.4,
     * '  AREA=',F10.4)
      STOP
      END
      SUBROUTINE CALC(X,C,A)
      DATA PI/3.14159/
      C=X*PI
      A=(X/2)**2*PI
      RETURN
      END
/DATA
3.51
*End
*Go

fdebug.sample
*In progress

**MAIN** END OF COMPILATION 1 ******

**CALC** END OF COMPILATION 2 ******
004F50 BYTES USED
EXECUTION BEGINS
VS FORTRAN VERSION 2 RELEASE 4 INTERACTIVE DEBUG
5668-806 (C) COPYRIGHT IBM CORP. 1985, 1989
LICENSED MATERIALS-PROPERTY OF IBM
WHERE: MAIN.1
FORTIAD
?
listsubs
PROGRAM UNIT                     COMPILER   OPT   HOOKED TIMING
MAIN                            VSF 2.4.0    0      YES    OFF
CALC                            VSF 2.4.0    0      YES    OFF
FORTIAD
?
at calc.4
FORTIAD
?
go
AT: CALC.4
FORTIAD
?
list (calc.x,calc.c,calc.a)
CALC.X           =  3.51000023
CALC.C           =  11.0269814
CALC.A           = 0.000000000E+00
FORTIAD
?
go
DIAMETER=   3.510  CIRCUMFERENCE=   11.0270  AREA=    9.6762
PROGRAM HAS TERMINATED; RC ( 0)
```

```
FORTIAD
?
listfreq
STATEMENT                                          FREQUENCY
MAIN.ENTRY                                          NO HOOK
MAIN.EXIT                                           NO HOOK
MAIN.1                                              1
MAIN.2                                              1
MAIN.3                                              1
MAIN.5                                              1
MAIN.6                                              0
FORTIAD
?
quit
*End
*Go
```

Separation Tool - VSFORT

The VS Fortran Separation Tool utility program is used to create the re-entrant and non-reentrant parts of a VS Fortran program compiled with the RENT option. Refer to the VS Fortran Programming Guide. The re-entrant part of a program could be put into MUSIC's Link Pack Area to provide better performance for high use programs.

Control statements for running the Separation Tool are:

```
/PARM name              <--- for "assigned name" form only
/FILE SYSUT1 NAME(nonrent.obj) NEW
/FILE SYSUT2 NAME(rent.obj) NEW
/INCLUDE VSFORT.SEPTOOL
/INCLUDE prog.obj       <--- object from compile with RENT
/INCLUDE sub1.obj
...
```

Alternate Math Library (Version 1 Only)

To use the Alternate Math Subroutine Library supplied with VS Fortran Version 1, insert the following statement after your Fortran source (or after your object modules when creating a load module):

```
/INCLUDE MATH.ALTLIB
```

Sample Program - VSFORT

```
list vsfort.sample
*In progress
/SYS REGION=512
/LOAD VSFORT
C  VS FORTRAN SAMPLE PROGRAM.
C  CALCULATES RADIUS, CIRCUMFERENCE, AND AREA OF A CIRCLE,
C  GIVEN THE DIAMETER.
```

```
C  ANSWERS CAN BE REQUESTED IN HIGH OR LOW PRECISION.
C  INPUT IS BY MUSIC NAMELIST.
      REAL*8 DIAM,RADIUS,CIRCUM,AREA
      LOGICAL LOW,HIGH
      NAMELIST /DATAIN/ DIAM,LOW,HIGH
100   WRITE(6,*) 'Enter: DIAM=value,LOW  OR  DIAM=value,HIGH'
      LOW=.FALSE.
      HIGH=.FALSE.
      READ(9,DATAIN,END=200)
      IF(.NOT.(LOW.OR.HIGH)) LOW=.TRUE.
      CALL CALC(DIAM,RADIUS,CIRCUM,AREA)
      IF(LOW) WRITE(6,10) DIAM,RADIUS,CIRCUM,AREA
10    FORMAT(' DIAMETER=     ',F9.4,3X,'RADIUS=    ',F9.4/
     *       ' CIRCUMFERENCE=',F9.4,3X,'AREA=      ',F9.4)
      IF(HIGH) WRITE(6,20) DIAM,RADIUS,CIRCUM,AREA
20    FORMAT(' DIAMETER=     ',F12.6,3X,'RADIUS=    ',F12.6/
     *       ' CIRCUMFERENCE=',F12.6,3X,'AREA=      ',F12.6)
      GO TO 100
200   STOP
      END
      SUBROUTINE CALC(X,R,C,A)
      REAL*8 X,R,C,A,PI/3.14159265/
      R=X/2
      C=X*PI
      A=R**2*PI
      RETURN
      END
/INCLUDE MUSNL.OBJ
*End
*Go

vsfort.sample
*In progress

**MAIN** END OF COMPILATION 1 ******

**CALC** END OF COMPILATION 2 ******
006010 BYTES USED
EXECUTION BEGINS
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
diam=3.51,low
DIAMETER=        3.5100   RADIUS=       1.7550
CIRCUMFERENCE=  11.0270   AREA=         9.6762
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
diam=3.51,high
DIAMETER=         3.510000   RADIUS=        1.755000
CIRCUMFERENCE=   11.026991   AREA=          9.676185
Enter: DIAM=value,LOW  OR  DIAM=value,HIGH
?
/eof
*End
*Go
```

References - VSFORT

Manuals for VS Fortran Version 1

VS Fortran Language and Library Reference, SC26-4119.

VS Fortran Programming Guide, SC26-4118.

Manuals for VS Fortran Version 2

VS Fortran Version 2 Language and Library Reference, SC26-4221.

VS Fortran Version 2 Programming Guide, SC26-4222.

VS Fortran Version 2 Interactive Debug Guide and Reference, SC26-4223.

VS Fortran Version 2 Master Index and Glossary, SC26-4603.

VS Fortran Version 2 Reference Summary, SX26-3751 (booklet).

MUSIC/SP VS/FORTRAN Debugger - TESTF

TESTF, the VS/FORTRAN Full Screen Debugger, enables you to control your VS/FORTRAN program completely during the test and debugging stages of program development. You can set break-points, inspect and alter variable values, scan the source for text, generate a trace listing (which includes actual source statements), re-start execution at another instruction (within the current module), cause execution to cease at entry and/or exit of routines, and manipulate arrays. As well, TESTF is integrated with DEBUG, the machine level debugger. This enables you to view and execute the actual machine instructions that implement your VS/FORTRAN instructions. Of course, MUSIC commands can be issued from TESTF.

Invoking TESTF (VS Fortran Debugger)

The easiest way of invoking TESTF is to use the TESTF REXX exec. A simple example: in order to test your VS/FORTRAN program, called ASSIGN1, enter the following:

```
testf  assign1
```

```
---- VS/FORTRAN Interactive Debug ---- Where: MAIN     @ ISN:       6  Line:    7
Command ==>
Data Window Displayed
--ISN--*....*...1.........2.........3.........4.........5.........6.........7..
    2        INTEGER*2    I2(2,2,2,2)
    3        REAL         X(4,4)
    4        INTEGER I1,II1,III1,J,K,L
    5        LOGICAL  A,B,C
       C
    6  10    A = .TRUE.
    7        B = (.NOT.A)
    8        C = .FALSE.
       C
    9        I1 = 10
   10        II1 = 100
Module---Variable-Address-Type-----Len-Atr--Hex Value--Value by Type ----------
MAIN     II1      004990  INTEGER4  4       00000000    0
MAIN     III1     004994  INTEGER4  4       00000000    0
MAIN     CHRVAR   005800  CHAR     50    A  E3C8C9E240  THIS IS THE  (1,1) ELEME

------------------------------------------------------------------- ----------
1:Help   2:Run Pgm        5:Single Step    7:Page Up    4:Arrays    10:Debug
3:End    11:Animate Pgm   9:Break Point    8:Page Down  6:Data      12:Retrieve
```

Figure 8.6 - Example of MUSIC's VS/FORTRAN Debugger Display

The following is a more detailed description of the TESTF exec.
The complete format for invoking TESTF is as follows:

```
TESTF source OBJECT(...) PARMS(...) REGION(...) FILES(...) LINK
             LISTING(name) DATA(name)
```

where the above parameters have the following interpretation:

source-name	name of source file to compile and test; if omitted then the exec assumes that you wish to run from object modules (using the OBJECT parameter)
OBJECT(...)	name of object file to be included at run or link edit time; can contain /INCLUDE statements
LINK	requests that a load module be generated and that it subsequently be executed
FILES(...)	name of a model file containing /FILE statements that will be used when the program is executed.
LISTING(...)	name of a listing file if no source file was provided. The file specified will be used for the listing if a source file was specified.
REGION(...)	specifies the region size to be used for all jobs
PARMS(...)	application parameters for your VS/FORTRAN program
DATA(name)	specifies the name of a data file that will be accessible via READs on unit 5. Note that this file should already exist; otherwise, an error will result.

If you are not using the TESTF exec, then you should be aware of the following. Modules to be debugged MUST be compiled with the TEST and SDUMP compiler options. You must request a source listing, to be saved in a file with LRECL 133. In order to use the Debugger, set up JCL as in one of the following 3 examples (compile and go, load-and-go, and exec from load module).

Note that the VS/FORTRAN Debugger first looks for a DDname DBGLIST for the source listing, and then (if not found) searches for SYSPRINT. So if there is a conflict with using SYSPRINT, then utilize DBGLIST instead.

Also, note that in the case of a large application, not all of the application need be debugged. You can elect to debug only certain modules; the Debugger will "wake up" within these modules. However, you must ensure that the VS/FORTRAN environment has been established; this requires that you have a MAIN program or at least call the VS/FORTRAN initialization routines. Refer to the VS/FORTRAN Programmer's Guide for more information on how to call VS/FORTRAN initialization routines, if you do not have a VS/FORTRAN main program.

Compile-and-go

```
/SYS REG=1500
/FILE SUBLIB  PDS(*DBG,*OS,*MUS)
/FILE SYSPRINT N(module.LST) LR(133) NEW(REPL)
/LOAD VSFORT
/OPT SOURCE,SDUMP,TEST
/INCLUDE module.S
```

Load-and-Go

```
/SYS REG=1500
/FILE SUBLIB  PDS(*DBG,*OS,*MUS)
/FILE SYSPRINT NAME(module.LST) SHR
/LOAD ASMLG
/INCLUDE module.S
```

Load Module Exec

```
1) Linking

   /FILE LMOD NAME(module.LMOD) NEW(REPL) LR(128) RECFM(F)
   /FILE SUBLIBOS  PDS(*DBG,*OS,*MUS)
   /LOAD LKED
   /JOB NOGO,MODE=OS,NAME=module
   /INCLUDE module.obj
2) Execution

   /SYS REG=1500
   /FILE SYSPRINT NAME(module.LST)  SHR
   /FILE LMOD NAME(module.LMOD)  SHR
   /LOAD XMON
   module
```

VS Fortran Debugger Function Keys

F1 Help — Accesses the help text.

F2 Run — This key/command causes your program to resume executing, without returning control to the Debugger unless a breakpoint is encountered, an error occurs, or your program ends. Any breakpoints that have been set will be honoured, as will Entry and Exit stops (set via the Halt command). When the program terminates a message to this effect is placed in the message area.

If you have enabled tracing with the TRACE command, then a trace listing will be produced as execution proceeds.

F3 Exit — This key/command causes an immediate exit from both your program and the Debugger.

F4 Arrays — Pressing this key presents the Array display. The Array display allows you to display and modify values of an array. Placing the cursor on an array in the data display causes that array to be displayed.

In the Array display, you can modify the index values that appear on the same line as the array name, in order to display that array element as the first line. As you scroll up and down through the array, the index values are altered in order to display the index value for the top-most array element in the Array display. (Keep in mind that first index value changes the fastest, and so on; index values are in the same order as would be used in your program.)

F5 Single Step — Control is advanced to the next debuggable statement. Normally, this means that a single statement is executed, but when a routine is called that has not been compiled with the TEST option, the Debugger cannot step into the routine; thus the whole routine is executed. Any variables that have changed are updated on the screen.

If you place the cursor on a line and press F5 (STEP), then a temporary breakpoint is placed on that line, instead of the next instruction, and execution resumes, until that temporary breakpoint is encountered. In other words, your program will be executed up until that line.

A typical use of this feature is to step past a DO loop without having to set a breakpoint, then issuing the RUN command, and then resetting the breakpoint. Note that when using this feature, there is no guarantee that control will actually be returned to you at that statement. If the logic of your program is such that control never arrives at that statement, then the use of this feature is equivalent to a RUN command. If you have no other breakpoints, then your program will run to termination.

F6 Data

Pressing this key presents the Data display. The Data display allows you to display and modify values of variables known to the Debugger. Use the FIND VARIABLE (FV) command in order to locate a variable, and use the SHOW and HIDE commands in order to control which variables are displayed. Use options in the SET command as well in order to control how and when the variable list is modified.

In the Data display, you can modify a variable in both hexadecimal and "natural" type. In other words, if the variable is of character type, you can either type in hexadecimal codes for the characters, or the characters themselves.

F7 Up
F8 Down

Causes the either the Source Window or the data window to scroll up or down by one page. The Data Window is either the Data, Array, or Watch display. The Source Window contains the source lines being debugged. The window that is scrolled depends upon where the cursor is located. If the cursor is in the Source Window, then the Source Window is scrolled (note: the command line is considered part of the source window). If the cursor is in the Data Window, then the Data Window is scrolled.

If the cursor is in the Source Window on a source line, pressing F7 or F8 causes that line to be positioned at the bottom or top of the Source Window Display, respectively. The cursor is placed into the command area. This allows you to "fine tune" your position in the source listing, without having to use the LINE command.

F9 Set/Reset Break Points

Placing the cursor on a source line with an ISN (Internal Sequence Number) sets a break point at that line, indicated by the characters 'ISN' being replaced by 'Brk'. Repeating this caused the break point to be reset.

F10 Debug

This places you into Debug (machine level), all set up to Debug the machine language instructions for the current VS/FORTRAN instruction. To return to the VS/FORTRAN Debugger, press F2 (RUN) or F3.

The machine level Debug is integrated with the VS/FORTRAN Debugger, in that it understands the VS/FORTRAN TEST environment. The machine level debugger displays the current VS/FORTRAN instruction being executed in the message area. Pressing F3 returns to the VS/FORTRAN Debugger without executing the instruction, whereas F2 executes the machine instructions for that statement, and then returns to the VS/FORTRAN debugger.

Be careful not to alter 'Set System Mask (SSM)' instructions, as they provide ISN information to the Debugger. When you reach a BAL instruction in front of one of these, press F5 (Step). This will position you at the next instruction to be executed (The corresponding VS/FORTRAN statement will be updated, as well).

F11 Animated Execution

This key causes the Debugger to go into animated execution mode. Approximately once per second (default value - "Set Delay" can alter this), an instruction is executed and the screen is updated. The effect is that of continuously single-stepping

your program. In order to exit this mode, press Enter and animation will cease.

F12 Retrieve — This key retrieves previously entered commands into the command field. This allows you to re-issue commands after (possibly) correcting them, for example.

VS Fortran Debugger Commands

Although only the full form of the command name is given here, all commands can be specified with the minimum number of characters necessary to unambiguously identify the command.

Some commands refer to the 'current module'. This means the module of your program that is currently executing. You can determine what this is by looking on the first line of the screen; the name after the label WHERE is the name of the 'current module'.

Find

The Find command has a number of variants. Although the actual syntax is:

```
find search-type  arg
```

where 'search-type' is the function being requested, you can specify each variant of the FIND command by 2 characters, given below.

Find String (FS) — This command locates a string in the source listing, starting from the current line. If the string is not found, then the search is re-started from the top. Thus the effect is that the string is found, no matter where it is in the source; the search "wraps around", so to speak.

Syntax:

```
find string text          or     fs text
```

where 'text' is the string to be located. the current line.

Find Variable (FV) — The FIND VARIABLE command locates a variable in the variable list, starting from the first variable displayed. As with the string search, the search "wraps around" if the variable is not found between the current line and the bottom of the list. If a variable is hidden, it is made visible if it is found.

Note that by default, the search is restricted only to the current module. In order to search the entire variable list, use the '*' option for the module name.

Syntax:

```
find var name             or     fv name
find var * name           or     fv * name
find var mod name         or     fv mod name
```

where 'name' is the name of the variable to be located, 'mod' is the module name to which the search should be restricted, and '*' means locate a variable with the specified name, regardless of which module it is in.

Find Module (FM) — This command searches for the named module. In addition to subroutines which

have explicit names, the name MAIN is recognized as the name of the main program, even though it does not appear in the source. This is because the command operates on the results of an analysis of the source input. By the same token, any module not in the source listing cannot be found.

Syntax:

```
find module mod          or     fm mod
```

where 'mod' is the name of the module to be located.

Find Breakpoint (FB)

This command finds the next breakpoint in the source listing. If a breakpoint is not found between the current line and the end, then the search "wraps around" from the start of the source listing.

Syntax:

```
find break               or     fb
```

Find Label (FL)

This command locates the statement label specified. Unless modified by a module name, the search is restricted to the current module.

Syntax:

```
find label  #            or     fl #
find label  *  #         or     fl *   #
find label mod #         or     fl mod #
```

where '#' is the statement label, 'mod' is the name of a module, and '*' means all modules.

Find ISN (FI)

This command locates the Internal Sequence Number (ISN) specified. (An ISN is the number assigned to all FORTRAN statements, except comments, and appears to the left of the FORTRAN statement in the listing.) Unless modified by a module name, the search is restricted to the current module.

Syntax:

```
find isn  #            or     fi #
find isn  *  #         or     fi *   #
find isn mod #         or     fi mod #
```

where '#' is the ISN, 'mod' is the name of a module, and '*' means all modules.

Line

This command allows you to specify exactly which line to display. Although the line numbers are NOT displayed on the screen, the line number of the top line of the current display is located in the upper right-hand corner of the screen. This command is intended as a convenience in moving quickly to a particular area of the source listing.

Top/Bottom

'TOP' moves the source window to the top of the source listing. 'BOTTOM' moves the source window to the end of the source listing.

Trace

This command controls whether tracing information is produced when your application program is running. Initially, tracing information is not produced. When TRACE is entered, it causes trace messages to be produced when you issue a RUN command or a STEP command (over more than 1 statement); this mode has been "enabled". (You can save these messages by issuing the MUSIC command /REC NEW before running TESTF, and issuing /REC OFF after you exit from TESTF. The trace messages will be in a file called @REC.000) Entering TRACE again causes this to be cancelled, or "disabled". The messages produced by this command tell you whether TRACE mode has been "enabled" (activated) or "disabled" (deactivated).

Note: Enabling TRACE, then entering the machine Debugger and tracing instructions there, causes the FORTRAN and machine trace outputs to be merged.

Halt

This command allows you to run without breakpoints and to interrupt the execution of your program at the entry or exit from subroutines.

no parameter	Entering HALT without a parameter displays the status of the HALT conditions, both ENTRY and EXIT (see below).
ENTRY	'HALT ENTRY' causes the Debugger to interrupt your program whenever control enters a subroutine. In other words, you receive control in the Debugger at the beginning of every subroutine.
EXIT	'HALT EXIT' causes the Debugger to interrupt your program whenever control exits a subroutine. In other words, you receive control in the Debugger at the end of every subroutine.
OFF	'HALT OFF' resets both Halt conditions to inactive.

Syntax:

```
halt
halt entry
halt exit
halt entry exit
halt off
```

SET

This command allows you to tailor how the Debugger operates. Typically, these options control default actions of the Debugger.

```
Set  AutoShow
     AutoHide
     Bell
     Count
```

```
Delay
```

AutoShow 'SET AUTOSHOW' allows you to alternate between two modes of presenting variables.

The default mode is that only variables in the current module are displayed. When you call another module, the list of variables is cleared and only the variables in that module are displayed. This guarantees that the value in the variable 'I', for example, that you see refers to the variable I in the module currently being executed.

The other operating mode does NOT reset the variable list every time a subroutine is called. This allows you by default to retain all variables in the list of variables. Normally, this is not necessary or desirable.

AutoHide This option affects how the SHOW command (see below) operates. By default, when a SHOW command is issued, the list of variables is first cleared. 'SET AUTOHIDE' alternates between this mode of operation and the other mode whereby the list of variables is not cleared by default (you can do this via the HIDE *.* command, see below).

Thus to view only 2 or 3 variables constantly, issue the 'SET AUTOHIDE' command. Then issue 'HIDE *.*', and finally, issue your SHOW command (for example, SHOW I J K). This will fix I, J, and K in the variable window. If you wish these to be preserved across subroutine calls, then issue the 'SET AUTOSHOW' command as well.

Bell This option switches between never sounding the alarm and sounding the alarm when an error or unusual condition occurs.

Count This alters the maximum statement count (default 5000). When you run your program, the debugger counts the number of instructions executed. When this number is exceeded, the debugger interrupts your program and places a message on the display.

This feature allows you to run your program without fear that you will "lose control" and be caught in an loop. Likewise, if your program is looping, this feature allows you to discover where it is doing this.

The value specified with COUNT should be a positive number. To disable counting, specify a large positive number (i.e. 999999).

Delay This option alters the default delay time of 1 second between statements when using Animated Execution (see ANIMATE command). Specify the number of seconds to pause between VS/FORTRAN instructions; it must be a whole number bigger than 0.

SHOW

This command allows you to control the variables displayed in the Variable window. (See also SET AUTOSHOW and SET AUTOHIDE.) You can specify a list of individual variables and/or templates. The '*' used as a variable or module name means all variables or all modules. If used as part of a name (for example, I*) it means all variables or modules starting with 'I'.

Entering SHOW without parameters causes the Debugger to report the number of variables shown, and the total number of variables known.

Syntax:

```
       +-- var-name         displays variable in current module
       |   module.var-name  displays variable in indicated module
  Show |   *.var-name       displays indicated variable in all modules
       |   module.*         displays all variables in indicated module
       +-- *.*              displays all variables, all modules
```

HIDE

HIDE accepts the same format of parameters as does SHOW. However, variables that match the template are hidden from view, and not displayed.

Entering HIDE without parameters causes the Debugger to report the number of variables hidden, and the total number of variables known.

GO

This command allows you to alter the next instruction to be executed. Normally, FORTRAN programs execute instructions sequentially, unless the flow of control is modified by a loop, GO TO, or IF-THEN-ELSE construction. Within the Debugger, however, for testing purposes it is sometimes useful to re-do a number of statements (perhaps when erroneous input is recognized and corrected). The GO command allows you to execute any instruction within the CURRENTLY EXECUTING module ONLY. Thus you cannot suddenly execute a statement in some subroutine that is not active.

Note: GO does not cause execution to commence. Use the RUN command for this. This gives you a chance to verify that the statement to be executed next is in fact the one you wanted to be executed.

Syntax:

```
go #
```

where '#' is the ISN (Internal Sequence Number, given to the left of the statement in the source listing).

External Command

In order to execute a MUSIC command (say to Edit or View a file), preface the command in the command line with a leading '/'. Thus, to Edit your file called 'ASSIGN1', enter the following in the command line:

```
/edit assign1
```

When you have finished your Edit session, you will be returned to the Debugger.

General Purpose Simulation System - GPSS

General Purpose Simulation System (GPSS V Program Product 5734-XS2) is a digital simulation program for conducting evaluations and experiments of systems, methods, processors and designs. The program provides many facilities to simplify simulation programming and to produce output reports. The program is a modification and adaptation of GPSS V available on the 360-370 Operating System.

Restrictions - GPSS

The following special features of GPSS may give problems under MUSIC, and their use should be avoided if possible:

- Update
- Read/Save
- Jobtape
- Run Length
- Auxiliary Storage
- Load
- HELP routines

Usage - GPSS

GPSS is invoked by using the /LOAD GPSS statement in the input stream. The /LOAD is followed by an optional /OPT statement specifying the main storage option (A, B, or C) to be used, then by the GPSS program statements. If /OPT is omitted, storage option A is used.

```
/LOAD GPSS
/OPT x                  (where x is A, B, or C)
...GPSS program statements...
```

The ddnames which are automatically predefined by the MUSIC/GPSS interface are: DINPUT1 (input statements), DOUTPUT (printed output), and the temporary work files DINTERO, DSYMTAB, DINTWORK, DREPTGEN, and DXREFDS. Other ddnames, if needed, must be defined by /FILE statements. If the default space is not enough for a temporary work file, supply an overriding /FILE statement of the form:

```
/FILE ddname NAME(&&TEMP) NEW RECFM(V) SPACE(n)
```

Reference - GPSS

General Purpose Simulation System V User's Manual, (GH20-0851)

Example - GPSS

```
/LOAD GPSS
*     GPSS HARBOUR SIMULATION PROBLEM
      SIMULATE
      GENERATE    2,1
      QUEUE 1
      ENTER 11
      DEPART      1
      QUEUE 2
      ENTER 12
      DEPART      2
      ADVANCE     15,6
      LEAVE 12
      TRANSFER    .9,,LVE
      ADVANCE     2,1
LVE   LEAVE 11
      TABULATE    10
      TERMINATE   1
11    STORAGE     6
12    STORAGE     5
10    TABLE M1,0,10,20
      START 100
      END
```

Graphical Data Display Manager - GDDM

MUSIC supports the GDDM series of programs, providing presentation services in host computers. It drives displays, printers, plotters, and scanners, and includes several easy to use utilities for end-users. These utilities can be accessed via a full-screen menu.

As well, GDDM has a powerful and versatile programming interface. This means that from your high level language program (VS FORTRAN, PL/1, VS PASCAL, etc.) you can call GDDM subroutines and perform graphics function upon your workstation. Using PCLK, you can generate graphics on MUSIC/SP that can be displayed on your PC (when dialed into the MUSIC/SP host running on a 9370).

Command - GDDM

To start any of the GDDM functions (Presentation Graphics, View Utility) issue the command: GDDM. From the panel screen, select the option and press enter. In order to use the System Programming Interface of GDDM, refer to the GDDM reference manual for the subroutine calling sequences. You must use the appropriate subroutine library as described below.

You should specify a 3 megabyte user region when using GDDM.

Usage Notes - GDDM

If you are coding your own applications to use GDDM's System Programming Interface, then you will need to use the correct subroutine library when Linkediting the program.

If the program is using the GDDM reentrant interface, use the following subroutine library search order: (*GDDMRLIB, xxx, *GDDM, *OS, *MUS).

If your program is using the non-reentrant interface (this is usually the case), then use the following for the subroutine library search order: (*GDDMNLIB, xxx, *GDDM, *OS, *MUS)

In both of the above cases, replace "xxx" by *PGF, *IMD, *IVU when using the Presentation Graphics Facility, the Interactive Map Definition utility, or the Interactive View Utility, respectively.

For example, suppose that you are using the non-reentrant version of the System Programming Interface, in conjunction with the Presentation Graphics Facility. In generating a load module using the Linkage Editor, the required SUBLIBOS file statement would be:

```
/FILE SUBLIBOS PDS(*GDDMNLIB,*PGF,*GDDM,*OS,*MUS) SHR
```

placed prior to the /LOAD LKED statement.

Access to GDDM on MUSIC/SP via FTTERM V2, via the 9370 ASCII subsystem is supported. Consult the FTTERM documentation for further information.

Restrictions

1. You cannot use the MUSIC Loader with any GDDM applications. The MUSIC Linkage Editor must be utilized. This means that you must specify /JOB NOGO,DECK for the assembly or compilation step, and run a second job that performs the linkedit. (Refer to the Usage Notes above to see how to specify the GDDM subroutine libraries.)

2. If you are using the System Programming Interface of GDDM, then you MUST add the following call BEFORE attempting to call any GDDM facilities.

   CALL GDDMSB

   This call sets up the MUSIC-GDDM interface and then returns to the caller.

3. The sample User Tasking application requires a large amount of storage. Based on the order that you select the start-up, you can receive the MUSIC message about Insufficient Main Storage.

4. Depending on the type of workstation that you are using, the MUSIC multi-session facility may function a little differently. When you press the PA2 key, you MAY get a "X PROG752" message at the bottom of your workstation. This is caused by the fact that GDDM has set the workstation in 16 bit addressing mode. Multi-session is STILL available. You will NOT receive the message "Press function key to go to next session". To use multi-session, you must:
   1. Press RESET on the workstation.
   2. Press the function key for the multi-session function you wish to perform.

Full-Screen Menu - GDDM

The following menu is presented when you issue the GDDM command from the *Go mode. This front-end menu allows you to start up the various GDDM utilities without having to remember the actual utility name.

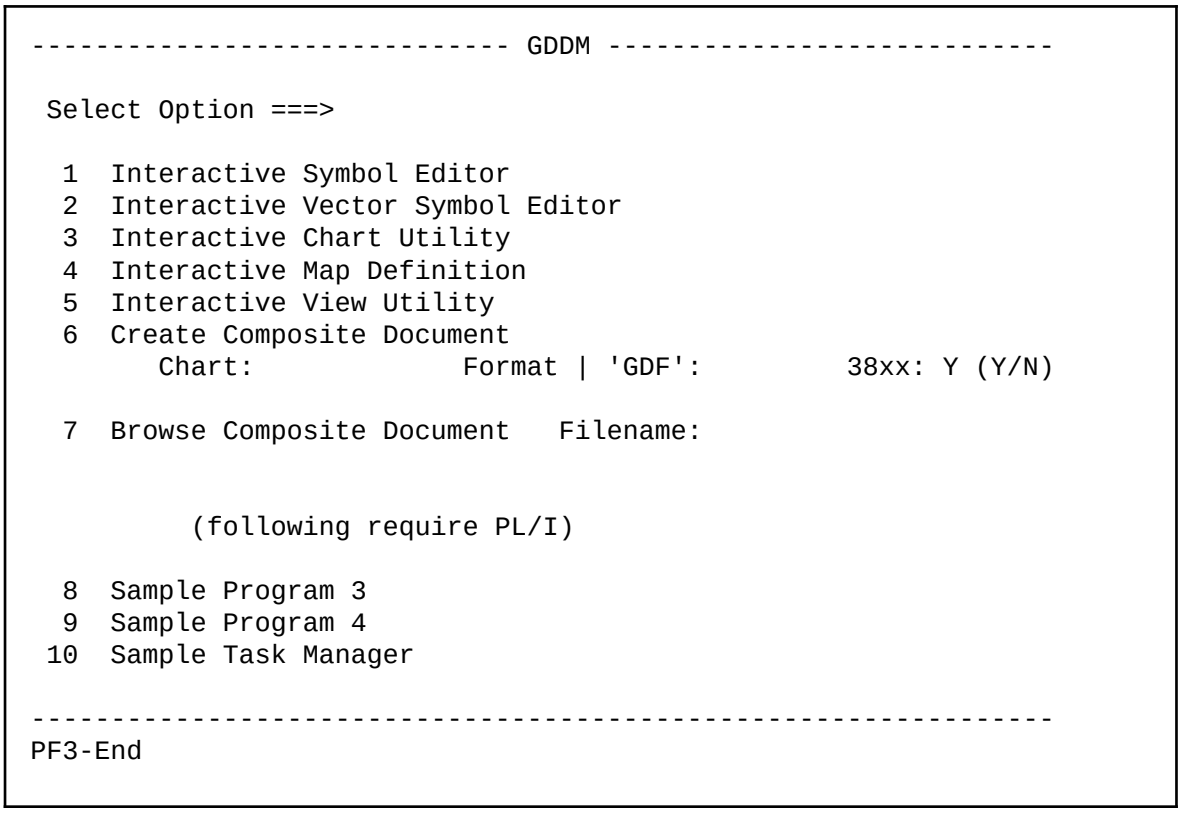

Figure 8.7 - Menu for GDDM

Two symbol editors (the Image Symbol Editor and the Vector Symbol Editor) allow you produce or modify symbols for annotating graphical output.

The Interactive Chart Utility helps you to draw charts in a simple manner for display on a screen or for printing. No programming knowledge is required in order to run the Interactive Chart Utility.

The Image Map Definition utility allows you to define the layout of the data that your application program presents on a display or output device. Basically, it allows you to define screens for your programs using GDDM.

The Interactive View Utility is an interactive program which allows users to create images by scanning documents, save images on disk, display them on screens, edit them in various ways, and create image output files for printers.

A composite document is a file containing text, images, and graphics. Option 6 on the GDDM menu allows you to create these documents and save them in MUSIC files. Option 7 enables you to view these documents on a workstation or a PC (using PCLK).

References - GDDM

There are a large number of manuals available for GDDM. Refer to the General Information Manual below for more information on the manuals that you should reference. A selection of some other GDDM manuals has also been included below.

GDDM General Information Manual, (GC33-0319).

GDDM Image View Utility, (SC33-0479).

GDDM Interactive Map Definition, (SC33-0338).

GDDM-PGF Interactive Chart Utility, (SC33-0328).

GDDM Image Symbol Editor, (SC33-0329).

GDDM-PGF Vector Symbol Editor, (SC33-0330).

GDDM Application Programming Guide, (SC33-0337).

GDDM-PGF Programming Reference, (SC33-0333).

GDDM PCLK Guide Version 1.1, (part number 6242915).

Examples - GDDM

The following samples show the MUSIC control statements needed to compile, linkedit and execute a GDDM sample program.

1. Compiling a GDDM Sample Program

```
/SYS REG=1000
/FILE SYSPUNCH NAME(PLI4.SAMPLE.OBJ) NEW(REPL)
/FILE SYSLIB PDS($GDM:*.M) SHR
/LOAD PLI
/JOB NOGO
/OPT INCLUDE,MARGINS(2,72,0),DECK
/INC $GDM:ADMUSP4.S
```

2. Link-Editing a GDDM Sample Program

```
/FILE LMOD NAME(PLI4.SAMPLE.LMOD) NEW(REPL) LR(128) RECFM(F)
/FILE SUBLIBOS PDS(*OS,*GDDMNLIB,*GDDM,*MUS) SHR
/LOAD LKED
/JOB NOGO,MODE=OS,MAP
/INC PLI4.SAMPLE.OBJ
```

3. Executing a GDDM Sample Program

```
/SYS REG=3000
/FILE ADMSYMBL PDS(*.SYM,$GDM:*.SYM) SHR
/FILE ADMDEFS N(GDDM.ADMDEFS) SHR
/FILE ADMGDF PDS(*.GDF) SHR
/COM if you have PC-Link then you need the next statement
/FILE ADMPC PDS($PLK:*.SYM) SHR
/FILE 3 UDS($GDMGDDM) VOL(MUSIC1) SHR
/FILE LMOD NAME(PLI4.SAMPLE.LMOD)
/LOAD XMPLI
TEMPNAME LMOD
```

Linkage Editor - LKED

The MUSIC Linkage Editor is a loader which is capable of creating a load module with an overlay structure, similar to the IBM Operating System Linkage Editor. (The load modules, though, are not interchangeable with OS.) If an appropriate /FILE statement for a UDS or file is provided, the load module will be written on it. Load module data sets should be allocated with a record size of 128 bytes, and record format F should be used for files. When loading is complete, the job is executed (unless /JOB NOGO is specified). The Linkage Editor overlay supervisor is part of the user program during execution and requires about 2000 bytes of main storage.

Since load modules in some cases contain system-dependent coding, users should be prepared to recreate them if required because of system modifications.

Notes:

1. The ALIAS, INCLUDE and LIBRARY Operating System Linkage Editor control statements are not supported.

2. The user should note that if an overlay segment is to be used more than once, it must be serially reusable. Serially reusable means that an overlayed subroutine does not assume that variables will still contain values set by some previous execution of that subroutine.

3. If a program to be overlaid uses VS FORTRAN direct access input/output, all DEFINE FILE statements must be in the root segment of the overlay.

Link-editing COBOL, VS Assembler and PL/I Programs

VS FORTRAN, COBOL, VS Assembler, and PL/I programs which perform input/output through BSAM or QSAM macro instructions must be link-edited with MODE=OS and NOGO specified on the /JOB statement. To run these programs from a load module, the user must use the /LOAD XMON statement (for COBOL, VS FORTRAN, and VS Assembler) or the /LOAD XMPLI statement (for PL/I) in place of the /LOAD EXEC statement.

Usage - LKED

The Linkage Editor is invoked by a /LOAD LKED statement. This statement may be followed by object modules and overlay control lines or by a /OPT SYSIN=n pointing to a data set containing object modules and overlay control statements.

The file to contain the load module is defined by a /FILE statement specifying ddname LMOD. This /FILE statement precedes the /LOAD LKED statement. Unit number 3 may be used instead of ddname LMOD. Either a file or a UDS may be used, as in the following examples:

```
/FILE LMOD NAME(filename) NEW LRECL(128) RECFM(F) SPACE(40)

/FILE LMOD UDS(dsname) VOL(MUSIC1) NEW LRECL(128) NREC(500)
```

To distinguish load module files from other files, it is a good idea to use a suffix such as .LMOD at the end of the file name.

Parameters - LKED

The /LOAD LKED statement may be followed by a /JOB line indicating optional parameters.

```
/JOB  [NOMAP][,NOCALL][,NONOPRINT][,NOGO][,SEQUENCE][,TRACE]
      [MAP  ][,CALL  ][,NOPRINT  ][,GO  ]
                      [,PRINT    ]

           [,NAME=prgnm][,MODE=OS][,LET][,STATS][,MAXGAP=n]
```

NOMAP No module map to be produced. This is the default for workstation jobs.

MAP A module map showing each csect and entry point with its assigned relative address is to be printed. This is the default for batch jobs. Absolute addresses can be found by adding hexadecimal 4A00 to the relative addresses.

NOCALL Do not search the system subroutine library for required modules not present in the input stream. The parameter NOSEARCH is equivalent to NOCALL.

CALL Search the system subroutine library for required modules not present in the input stream. This is the default. The parameter SEARCH is equivalent to CALL.

NONOPRINT Suppress printing of all control statements and error messages.

NOPRINT Suppress printing of all control statements. This is the default for workstation jobs.

NOLIST Same as NOPRINT.

PRINT Print all control statements and error messages. This is the default for batch jobs.

LIST Same as PRINT.

NOGO Do not execute the load module produced.

GO Execute the load module only if no errors were detected during Linkage Editing.

LET Execute the load module even if errors were detected during Linkage Editing. This is the default.

SEQUENCE Check sequence numbers in columns 77-80 of all input object modules to detect any missing or out-of-sequence lines. Blank sequence numbers are ignored.

TRACE Display a message identifying each csect as it is read by the Linkage Editor.

MAXGAP=n Specifies that any uninitialized storage areas of length greater than *n* bytes are to be represented as a gap in the load module text on disk. The default is MAXGAP=8192.

The following parameters are only meaningful if the load module is being created on a file that is to be kept for later use.

NAME=prgnm Specifies the name of the load module, and is required. *prgnm* can be up to 8 characters in length and must be alphanumeric with no special characters. The Linkage Editor control

statement NAME will override this option. If not specified, NAME=TEMPNAME is assumed.

STATS — Specifies that the Linkage Editor is to display a message showing the total number of blocks in the load module data set, and the number of blocks used for this load module. A *block* is a 512 byte piece of the file. Thus the file statement "/FILE LMOD UDS(&&TEMP) NREC(400) RSIZ(128)" contains 100 blocks.

MODE=OS — Specified if link-editing COBOL, PL/I or VS Assembler programs as described above. The NOGO option must also be specified.

Notes:

1. If a /JOB statement is used, it should appear immediately after the /LOAD LKED statement. More than one /JOB line may be used if necessary, but each parameter must be wholly contained on one /JOB line.

2. For a very large program, the Linkage Editor may stop with an error message such as `too many RLDs` or `too much text.` In that case, try the Linkage Editor job again in a larger user region, e.g. /SYS REGION=1024. A larger work file may also be needed (see note 3 below).

3. The Linkage Editor uses a temporary UDS with a DDNAME of LKEDWORK as a work file. This UDS has a default size of 8000 128-byte records. If needed, the user can increase this size by specifying a permanent UDS (which is deleted at the end of the job) by using the appropriate /FILE statement. For example:

```
/FILE LKEDWORK UDS(dsname) NREC(20000)
/ETC LRECL(128) VOL(MUSIC1) NEW DELETE
```

4. The following statement is needed when you use /LOAD LKED to execute a COBOL2 load module:

```
/FILE SUBLIBOS PDS(*COBOL2,*MUS,*OS,*EXT)
```

Control Statements - LKED

The Linkage Editor control statements normally begin at or after column 2. These statements should be preceded by all object modules (or /INCLUDE statements pointing to object module files).

Note that some Linkage Editor control statements have names which are similar to MUSIC's statements though they are NOT the same. (The MUSIC Linkage Editor takes its control statement names from those of the OS Linkage Editor.)

The following Operating System Linkage Editor control statements are supported on the MUSIC Linkage Editor:

OVERLAY
INSERT
ENTRY
NAME
REPLACE
CHANGE

The following control statements are not supported:

ALIAS
INCLUDE
LIBRARY

The /OPT SYSIN=n control statement may be used to read the input from MUSIC unit number n. Normally this control statement is used when you want to read object modules from a UDS file. When end-of-file or a /DATA statement is reached on the data set the Linkage Editor switches back to the normal input stream for further input.

Linkage Editor Return Codes

The following job exit codes (return codes) are set by the Linkage Editor:

0 No errors or warnings.
4 Some warning messages were issued. For example, there were some unresolved external references.
8 Some error messages were issued.
16 The Linkage Editor job was aborted because of a serious error.
>16 Some other system error.

If more than one load module member is created in the job, i.e. more than one NAME statement was used, the exit code is the highest code encountered for all the members.

If /JOB NOGO is not used, and the user's program is executed as part of the Linkage Editor job, the exit code is as set by the user program, rather than as above.

Reference - LKED

IBM 360/370 Operating System Linkage Editor and Loader (GC28-6538)

Example - LKED

```
/FILE LMOD NAME(PROG2.LMOD) NEW LRECL(128) RECFM(F)
/LOAD LKED
/JOB MAP,NAME=TEST2
/INCLUDE MAIN     (FILE CONTAINING THE MAIN PROGRAM)
/INCLUDE SUBS     (FILE CONTAINING THE THREE SUBROUTINES)
    OVERLAY A
    INSERT SUB1
    OVERLAY A
    INSERT SUB2
    OVERLAY A
    INSERT SUB3
/DATA
    - USER DATA -
```

Please refer to the example in the description of "EXEC - LOAD MODULE EXECUTOR" in this chapter for the control statements used to execute the load module created above.

Load Module Executor - EXEC

The Load Module Executor is used to load and execute a program which has previously been stored in a file by the MUSIC Linkage Editor. The executor is a part of the user program during execution, and requires about 2000 bytes of main storage. This loading process can be considerably faster than the use of the standard loader.

Since load modules in some cases contain system-dependent coding, users should be prepared to recreate them if required because of system modifications.

COBOL, ASM and PL/I Load Modules

The /LOAD EXEC processor automatically brings in the FORTRAN G interface. You must use the /LOAD XMON procedure if the load module is produced by an object module from some other compiler. For a load module produced by an object module from a PL/I program, use the /LOAD XMPLI procedure. The OS/MUSIC interface will be loaded in these cases.

Usage - EXEC

The Load Module Executor is invoked by a /LOAD EXEC statement. /LOAD EXEC specifies that a load module contained in a file is to be executed. /LOAD statements must not be used following this statement.

The /LOAD EXEC statement must be preceded by a /FILE statement specifying the file which contains the program, normally with a ddname of LMOD.

Control Line - EXEC

The Load Module Executor requires a single control line, which must be placed immediately after the /LOAD EXEC statement. This line is followed immediately by any user data required. (The /DATA statement is not used).

The format of the control line is as follows:

```
membername      ddname
                 or
                unit number
```

The member name must start in column 1, and is the name specified in the NAME=xxx option or on the NAME control statement when the load module was created by /LOAD LKED.

The second item is the ddname or unit number used on the load module /FILE statement in the /LOAD EXEC job. The default is ddname LMOD. It is separated from the membername by 1 or more blanks or commas.

Example - EXEC

Please refer to the example in the description of "LKED - LINKAGE EDITOR" in this chapter for the control statements used to create the load module referred in the following example. The load module was created in the file PROG2.LMOD, with module member name TEST2.

The following example shows how to cause this load module to be executed.

```
/FILE LMOD NAME(PROG2.LMOD)    (load module)
/LOAD EXEC
TEST2                          (member name of load module)
/INCLUDE DATA1                 (user's data)
```

Note: A /DATA statement does not precede user data.

Load Module Executor - XMON

The /LOAD XMON processor allows programs to be executed from load modules. This usually results in faster loading (compared with /LOAD ASMLG or /LOAD COBLG) and allows overlay structures to be used. The load module is created from the program object modules in the usual way, using the MUSIC Linkage Editor, except that the parameters *MODE=OS* and *NOGO* should be specified on the Linkage Editor /JOB statement. The load module is executed using /LOAD XMON, in much the same way as /LOAD EXEC is used. For /LOAD XMON, the user's load module is loaded at address 4A00 (hexadecimal).

Usage - XMON

```
/LOAD XMON
```

XMON is used as follows:

```
/FILE statements as required
/FILE LMOD NAME(filename) or /FILE LMOD UDS(dsname) VOL(volume)
/LOAD XMON
parameter statement (see below)
...user data if any...
```

The parameter statement (immediately following /LOAD XMON) has the following format:

membername	ddname or unit number	options

The member name must start in column 1, and is the name specified in the NAME= option or on the NAME control statement when the load module was created by /LOAD LKED.

The second item is the ddname or unit number used on the /FILE statement for the load module file. The default is ddname LMOD. This item must be specified if any additional options follow.

The following additional options, separated by blanks or commas, can be used:

SVCTRACE to trace supervisor call instructions in the program.

IOTRACE to trace input/output operations.

DUMP to produce a storage dump, in some cases, if the job abnormally terminates.

CDUMP conversational dump.

DEBUG invoke full-screen debugger.

NONRENT load a non-reentrant copy of OSTRAP into the user region. This enables you to test OSTRAP with DEBUG (requires that the debugged module reside in the user region).

LOADER - System Loader

The MUSIC loader may be used to load all programs except overlays and those programs requiring the OS/MUSIC interface. An input stream which consists entirely of object modules produced by VS Assembler will be processed more efficiently if the loader processor is invoked directly. After loading is complete, the program is executed (unless /JOB NOGO is specified).

Notes:

1. Object modules from high level language compilers and VS Assembler programs using BSAM or QSAM macro instructions cannot be executed using this loader. They require either /LOAD COBLG, /LOAD PLILG, or /LOAD ASMLG.

2. The loader can handle a maximum of 255 control section names and entry point names. If the combined total of CSECT names and entry names exceeds 255, use the linkage editor.

Usage - LOADER

The loader is invoked by the use of a /LOAD LOADER statement.

/LOAD LOADER specifies that the lines following consist exclusively of object modules to be loaded and executed by the MUSIC loader. Utilization of processing unit time is optimized if this statement is used instead of /LOAD BASIC or FORTG1. /LOAD statements following this statement are ignored.

Parameters: Loader Step - LOADER

```
/JOB  [GO  ][,MAP  ][,XESD=sss][,XRLD=rrr][,NOPRINT]
      [NOGO][,NOMAP]
```

GO	The program is to be loaded and executed. This is the default.
NOGO	The program is to be compiled only. (This option is often used when the DECK option on the /OPT statement is used from a workstation.)
MAP	A storage map showing where the user's programs and subprograms are loaded into main storage is to be produced.
NOMAP	No storage map is to be produced.
XESD=sss	The number of extra external symbol dictionary entries required (indicated when the loader is unable to load the program).
XRLD=rrr	The number of extra relocation dictionary entries required (indicated when the loader is unable to load the program).
NOPRINT	All loader messages, except error messages, are to be suppressed.

Note: If a /JOB statement is followed by another /JOB statement, only the last one is processed.

```
/OPT [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from MUSIC unit number n (1 through 15). A /FILE statement defining MUSIC I/O unit n as the appropriate file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream. If another /OPT SYSIN=n is found in the input stream, it will be processed in the same manner.

Example - LOADER

In the following example an object module was saved in a file called SAMPLE.OBJ. Edit the file SAMPLE.OBJ, and add the Job Control statement /LOAD LOADER as the first line. Then file the object module.

The example below shows the object module being executed.

```
*Go
sample.obj
*In progress
003638 BYTES USED
EXECUTION BEGINS
?
25.
THE ROOT OF  25.00 IS   5.00
?
/cancel
*Terminated
*Go
```

PASCAL Compiler - VSPASCAL

Programs written in the PASCAL language can be compiled and run using the IBM VS PASCAL compiler (Program Product 5668-767). Your installation may choose not to have this compiler available for your use.

VS PASCAL programs are normally compiled and executed using the OS/MUSIC interface. The VS PASCAL processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

Notes:

1. Since VS PASCAL programs can invoke many of the facilities of the Operating System, some of which are not available on MUSIC, it is possible to compile a VS PASCAL program which will not execute on MUSIC.

2. MUSIC supports fixed length sequential files. This includes VS PASCAL TEXT and RECORD file formats. VS PASCAL programs using the procedures PDSOUT and UPDATE (for sequential files) may be compiled but not run on MUSIC. Variable length records are supported only on output, and must not be blocked.

3. VS PASCAL direct access files using the SEEK procedure and fixed-length records are supported, provided that RESET or UPDATE procedure is used when opening the file. The REWRITE procedure must not be used for direct access files. For reading, open the file using the RESET procedure. For writing, use the UPDATE procedure. A new direct access file must be initialized in VS PASCAL by writing records **in order** (0,1,2,...) to the end of the dataset. Internally, each direct access record starts on a 512-byte block and occupies one or more blocks. To initialize a direct access file, use the REWRITE procedure to create a sequential file, writing valid or blank records in order to the file. Once the file has been created, it can be accessed as a direct access file.

4. If the VS PASCAL %INCLUDE statement is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) By default, the system provides a SYSLIB file statement pointing to the 'standard' VS PASCAL 'include library. If you wish to provide your own library and still have access to the 'standard' list, you must use the following SYSLIB statement.

   ```
   /FILE SYSLIB PDS('your library specifications',$VP2:*.M) SHR
   ```

   VS PASCAL supports two different forms of the %INCLUDE statement. Under MUSIC, only the following form is supported.

   ```
   %INCLUDE member-name
   ```

   The MUSIC /INCLUDE statement may also be used to perform a function similar to the %INCLUDE statement.

Usage - VSPASCAL

The VS PASCAL compiler is invoked by the use of a /LOAD VSPASCAL statement. /LOAD VSPASCAL specifies that the lines following consist of source statements to be processed by the IBM VS PASCAL compiler. Object modules may also be present. No other /LOAD statements should follow this one.

Main Storage Requirements - VSPASCAL

If the compiler is not in MUSIC's Link Pack Area, a region of at least 512K is required. If the compiler is in the Link Pack Area, a 256K region is large enough for small programs.

Interlanguage Communication - VSPASCAL

It is sometimes desirable to invoke subprograms written in other programming languages. A VS PASCAL procedure may call subroutines written in Assembler and FORTRAN. Subroutine written in COBOL or PL/I may not be called.

You may also invoke VS PASCAL procedures as subprograms from Assembler, COBOL and PL/I. When calling VS PASCAL subroutines from other languages, do not call the VS PASCAL procedure PSCLHX to cleanup the environment as documented. This procedure is not needed when running under MUSIC.

External File Names - VSPASCAL

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

DDNAME	LRECL	MUSIC I/O
INPUT	80	input stream (data following /DATA)
SYSPRINT	133	printed output
OUTPUT	133	printed output
SYSIN	80	conversational input (workstation job) input stream (batch jobs)

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

For any file, the LRECL and BLKSIZE may be specified in the VS PASCAL program by the LRECL and BLKSIZE options of the RESET, REWRITE and UPDATE procedures.

End-of-file can be indicated on a conversational read (of a TEXT file) by typing /EOF.

Parameters: Compiler Step - VSPASCAL

Compiler options may be specified on a /OPT statement preceding the VS PASCAL source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The default compiler options are listed below. Refer to the VS PASCAL Programmer's Guide for a description of the options and their abbreviations.

Workstation Defaults:	Batch Defaults (if different):
CHECK	
NODEBUG	
GOSTMT	
LANGLVL(EXTENDED)	
LINECOUNT(61)	
NOLIST	
MARGINS(1,72)	
OPTIMIZE	
PAGEWIDTH(128)	
PXREF	
SEQUENCE(73,80)	
NOSOURCE	SOURCE
NOXREF	XREF(SHORT)
DDNAME(COMPAT)	
HEADER	
CONDPARM()	
NOGRAPHIC	
STDFLAG(E)	

/FILE statements with ddnames of COMP.SYSIN, COMP.SYSPRINT and COMP.SYSLIN can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions.

Creating Object Modules

If you wish to compile a VS PASCAL procedure and save the object module for later execution, the following procedure should be followed.

1. Specify via the /FILE statement with ddname SYSLIN the name of the file to contain the object module.

2. Include the /JOB NOGO statement following the /LOAD statement. This will invoke the VS PASCAL compiler but will not executed the program after compilation.

VS PASCAL object modules can be executed using the OS-MODE loader, ASMLG. If a load module was created, then you must use the XMON procedure to execute the load module.

Parameters: Loader Step - VSPASCAL

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

          [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
DUMP	Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.
CDUMP	Request a conversational trace and dump when an error occurs.
DEBUG	Load the DEBUG program into memory for debugging.
FILRFM	Supply the RECFM V if the file is V/VC.
NOSEARCH	Do not search the subroutine library for unresolved routines.
IOTRACE	Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.
SVCTRACE	Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - VSPASCAL

Run time options can be specified on the "/PARM xxx" statement at the beginning of the job, where *xxx* is the option string. To distinguish run time options from the parameter string intended to be processed by the program, the options must proceed the parameter string (if any) and be terminated with a slash ("/").

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement. The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB Control statement above for a description of the syntax.)

Title Lines - VSPASCAL

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

VS PASCAL Interactive Debugger

The VS PASCAL interactive debugger is available under the MUSIC system. In order to use the debugger, you must follow the following steps:

1. Compile the module to be debugged with the DEBUG option. Modules that have been compiled with the DEBUG option can be loaded with modules that have not been compiled with the DEBUG option.

2. Specify on the /PARM statement, the DEBUG run-time option. After the modules have been loaded, the debug environment will be active and you will be immediately prompted for debugger commands.

3. The ATTN or BREAK keys on the workstation can be used to signal the debug environment that you want to gain control. On a 3270 workstation, after you press PA1 the message **ATTN** will be displayed in the lower right corner of the workstation. Press the ENTER key to signal the interrupt to the debug environment.

4. Use the VS PASCAL interactive debugger after creating a load module via the linkage Editor.

5. Use of the debugger environment will add about 70K to the User Region needed to execute the program.

References - VSPASCAL

VS PASCAL Application Programming Guide, (SC26-4319).

VS PASCAL Language Reference, (SC26-4320).

VS PASCAL Reference Summary, (SX26-3760).

Examples - VSPASCAL

1. The following sample program uses READ and WRITE statements on record files.

```
/LOAD VSPASCAL
program Sample;

type
  REC = record
          NAME : STRING(25);
          AGE  : 0..99;
          SEX  : (MALE,FEMALE)
        end;
var
  INFILE,
  OUTFILE:
      file of REC;
  BUFFER : REC;
begin
  RESET(INFILE);
  REWRITE(OUTFILE);
  while not EOF(INFILE) do
    begin
```

```
      READ(INFILE,BUFFER);
      WRITE(OUTFILE,BUFFER)
    end
end.
```

2. The following VS PASCAL program will be compiled and the interactive debug environment will be entered once the program has been loaded. Refer to the "Debug Terminal Session" in the *VS PASCAL Programmer's Guide* for an example of a debug session.

```
/SYS REG=340
/PARM DEBUG/
/LOAD VSPASCAL
/OPT DEBUG
program Debugging;
   var A, B : REAL;
begin
   A := 1;
   B := 2;
   Writeln ('A =     ', A);
   Writeln ('A + B = ', A+B);
end.
```

Note: The /OPT statement specifies the DEBUG compiler option. The /PARM statement specifies the DEBUG routine option. Both are required for DEBUG to be entered. In the /PARM statement, the trailing "/" serves to delimit VS PASCAL routine options from parameters that you might want to pass to your PASCAL program.

PL/I Version 2 Optimizing Compiler - PLI

Programs written in the PL/I language can be compiled and run using the IBM OS PL/I Version 2 Optimizing compiler (Program Product 5668-909). This compiler is compatible with previous releases of Version 2 and Version 1 (existing programs will work with this release).

The PL/I processor invokes the compiler, and then the loader (unless /JOB NOGO is specified). After loading, the program is automatically executed.

Usage Notes - PLI

Since PL/I programs can invoke many of the facilities of the MVS operating system, some of which are not available on MUSIC, it is possible to compile a PL/I program which will not execute on MUSIC.

1. MUSIC supports fixed length sequential files. This includes PL/I GET and PUT statements, and READ and WRITE statements for sequential files. The UPDATE and SEQUENTIAL attributes must not be used together. Variable length records are supported only on output, and must not be blocked.

2. Direct access files using REGIONAL(1) and fixed-length records are supported, provided that the UPDATE or INPUT attribute is used on the file declaration statement. The OUTPUT attribute must not be used. For reading, use the READ FILE statement. For writing, use the WRITE FILE or REWRITE FILE statement. A new direct access file should be initialized in PL/I by writing records **in order** (0,1,2,...) to the end of the data set. Internally, each direct access record starts a new 512-byte block, and occupies one or more blocks.

3. Sequential (ESDS), direct (RRDS), and indexed (KSDS) file access are available through MUSIC's VSAM support. For information on creating and using VSAM files with PL/I, refer to the topic "MUSIC/SP VSAM" in *Chapter 4. File System and I/O Interface*

   ISAM is not supported.

4. If the PL/I %INCLUDE statement is used, a /FILE statement with a ddname of SYSLIB and parameter PDS must be defined at the beginning of the job. (Consult the description of the /FILE statement for files.) The compiler option MACRO or INCLUDE must also be specified on the /OPT statement. The MUSIC /INCLUDE statement may also be used to perform a function similar to the %INCLUDE statement.

5. PL/I sort facilities are supported as described below. Extended precision arithmetic (up to 32 significant decimal digits) is supported.

6. The following are not supported:

   Multi-tasking.
   Teleprocessing (TRANSIENT file attribute).
   Execution-time COUNT, FLOW and REPORT options.
   Unbuffered attribute for RECORD files.
   Regional(2) and Regional(3) files

Usage - PLI

The OS PL/I Optimizing Compiler is invoked by the use of a /LOAD PLI statement. This processor invokes the PL/I Optimizing Compiler, and then the loader. After loading, the program is automatically executed using the OS/MUSIC interface (unless /JOB NOGO is used or the compiler return code is 12 or more). The /OPT and /JOB control statements may be used with this processor. PL/I object modules can optionally be produced and saved, intermixed with source. The object modules are identical with those produced on OS.

Available Unit Numbers and Buffer Space - PLI

Any tape blocksize up to 32760 may be used, provided the user region size (specified on /SYS REGION=nnn) is large enough.

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the compiler modules.

External File Names - PLI

A standard set of external file names (ddnames) is provided, along with default values for logical record length (LRECL).

DDNAME	LRECL	MUSIC I/O
SYSIN	80	input stream (data following /DATA)
SYSPRINT	121	printed output
SYSPUNCH	80	output to holding file (workstation jobs) punched output (batch jobs)
SYSTERM	121	printed output
CONSOLE	80	conversational input (workstation jobs) input stream (batch jobs)

Additional ddnames can be defined by specifying /FILE statements with the corresponding ddnames at the beginning of the job. (Consult the description of the /FILE statement.)

For any file, the LRECL and BLKSIZE may be specified in the PL/I program by the RECSIZE and BLKSIZE options of the ENVIRONMENT attribute.

File names within PL/I should not exceed 7 characters. However, the TITLE option on the OPEN statement may be used to associate a ddname with an internal file name. For example:

```
OPEN FILE(SYSPUN) TITLE('SYSPUNCH') OUTPUT;
```

When doing a conversational input on a workstation, always use PUT SKIP to force the output in the output buffer to be displayed on the workstation before the conversational read is performed. Otherwise, the prompts, if any, will not be synchronized with the conversational reads.

End-of-file can be indicated on a conversational read by typing /EOF.

Parameters: Compiler Step - PLI

Compiler options may be specified on a /OPT statement preceding the PL/I source. The options should be separated by commas, and the last option must be followed by a blank. A /OPT statement may also be used to indicate the start of a new external procedure. If more than one /OPT statement is used, the effect is cumulative, that is, each option remains in effect for the rest of the job until reset by a later /OPT statement.

The PL/I statement *PROCESS may be used in place of a /OPT statement. On a *PROCESS statement, options are separated by commas or blanks and the last option is followed by a semicolon. Options on *PROCESS statements are not cumulative.

The default compiler options are listed below. Refer to the *PL/I Optimizing Compiler Programmer's Guide* (*SC33-0006-5*) for a description of the options and their abbreviations.

Workstation Defaults:	Batch Defaults (if different):
GOSTMT	
GRAPHIC	NOGRAPHIC
LMESSAGE	
OBJECT	
STMT	
NOAGGREGATE	
NOATTRIBUTES	
NOCOUNT	
NODECK	
NOESD	
NOFLOW	
NOGONUMBER	
NOIMPRECISE	
NOINCLUDE	
NOINTERRUPT	
NOINSOURCE	INSOURCE
NOLIST	
NOMACRO	
NOMAP	
NOMARGINI	
NOMDECK	
NONEST	
NONUMBER	
NOOFFSET	
NOOPTIMIZE	
NOOPTIONS	OPTIONS
NOSEQUENCE	
NOSOURCE	SOURCE
NOSTORAGE	
NOTEST	
NOXREF	
NOCOMPILE(S)	
CMPAT(V2)	
FLAG(W)	FLAG(I)
LINECOUNT(55)	
MARGINS(1,80,0)	
SIZE(MAX)	
NOSYNTAX(S)	
SYSTEM(MVS)	

TERMINAL

/FILE statements with ddnames of PLI.SYSIN, PLI.SYSPRINT, and PLI.SYSPUNCH can be defined at the beginning of the job to inform the compiler where to read the input source program, where to print the program listing and punch the object module respectively, instead of the default definitions. To provide more space to the default compiler work file SYSUT1 (default space is 300K) for very large programs, supply the following /FILE statement before /LOAD PLI:

```
/FILE SYSUT1 NAME(&&TEMP) RECFM(F) LRECL(4096) SPACE(550) NEW DELETE
```

Parameters: Loader Step - PLI

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,DUMP][,CDUMP][,DEBUG]
      [FULMAP]

         [,FILRFM][,NOSEARCH][,IOTRACE[(C)]][,SVCTRACE[(C)]]
```

Parameters can be separated by one or more commas or blanks.

MAP	List a storage map of the loaded program.
FULMAP	List an extended storage map of the loaded program.
NOGO	Do not load or execute the program.
LET	Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.
NOPRINT	Informative and diagnostic messages are not to be produced.
DUMP	Request a storage dump to be produced if the job abnormally terminates. Even when the DUMP parameter is used, not all error conditions result in a dump.
CDUMP	Request a conversational trace and dump when an error occurs.
DEBUG	Load the DEBUG program into memory for debugging.
FILRFM	Supply the RECFM V if the file is V/VC.
NOSEARCH	Do not search the subroutine library for unresolved routines.
IOTRACE	Trace the I/O operations during the execution of the program. If IOTRACE(C) is specified, the trace for the I/O operations during the compilation of the program is performed.
SVCTRACE	Trace the SVC instructions during the execution of the program. If SVCTRACE(C) is specified, the trace for the SVC instructions during the compilation of the program is performed.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DUMP, IOTRACE and SVCTRACE options are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. other OS-mode processors that do not invoke the OS-mode loader directly.

Parameters: Go Step - PLI

Execution-time options can be specified by defining a PLIXOPT character string in the source program, or by specifying a "/PARM xxx" statement at the beginning of the job, where *xxx* is the option desired. The default execution-time options are:

HEAP(4K,4K)
ISAINC(0,0)
ISASIZE(x) where x=(region size - program size)/2
LANGUAGE(UENGLISH)
NOCOUNT
NOFLOW
NOREPORT
NOTEST
STAE
SPIE
TASKHEAP(4K,4K)

SVC instructions and I/O operations can be traced at execution time. This is done by specifying SVCTRACE and IOTRACE respectively on the /JOB statement The DUMP option can be specified on the /JOB statement to produce a storage dump, in some cases, if the job abnormally terminates. (Refer to the /JOB statement for a description of the syntax.)

Title Lines - PLI

Block letter titles (maximum 2 lines) can be printed on separator pages preceding the program listing if the job is run on batch. This is done by specifying a "=TITLE xxx" statement (for the first title line) and a "=TITLE2 xxx" statement (for the second title line) after the /LOAD statement. *xxx* is the title line and is from 1 to 10 characters in length.

PL/I Sort

A PL/I program can perform sorting operations by calling any of the sort interface subroutines PLISRTA, PLISRTB, PLISRTC and PLISRTD, as described in the PL/I Optimizing Compiler Programmer's Guide.

On MUSIC the sort is performed by a version of DSORT, the generalized sort routine described in *Chapter 10. Utilities* of this manual. The restrictions which apply to DSORT also apply to PL/I sorting. In particular, variable length records may not be sorted, sort control fields must begin on a byte boundary and be a whole number of bytes long, they may not overlap, and the maximum number of control fields is 20.

The argument specifying the amount of main storage for the sort must be at least 2048, and defaults to 32000 if not specified.

The following ddnames may be used. These ddnames, if required, must be defined using /FILE statements as described above in the section *External File Names*. An optional argument in the call to the sort interface routine may be used to change the first four characters *SORT* of the ddnames.

SORTIN Sort input if no E15 exit routine is defined.

SORTOUT Sort output if no E35 exit routine is defined.

SORTWK01 DSORT work file (optional).

SORTWK02 Second DSORT work file (optional). If SORTWK02 is not used, then SORTWK01 (if used) must be a UDS file, not a file.

If neither SORTWK01 nor SORTWK02 is defined by a /FILE statement, the sort must be small enough to be done entirely in main storage.

References - PLI

IBM OS PL/I Version 2 Programming Guide, Release 2, (SC26-4307).

IBM OS PL/I Version 2 Programming: *Language Reference*, Release 2, (SC26-4308).

IBM OS PL/I Version 2 Programming: *Messages and Codes*, Release 2, (SC26-4309).

IBM OS PL/I Version 2 Programming: *Using PLITEST*, Release 2, (SC26-4310).

IBM OS PL/I Version 2 Problem Determination, Release 2, (LY27-9528).

Example - PLI

This program reads numbers conversationally from the workstation and writes them to the data set FILE01. The end is indicated by typing /EOF. The object module is written to the file COPY.OBJ.

```
/FILE SYSPUNCH NAME(COPY.OBJ) NEW(REPLACE)
/FILE FILE01 UDS(USERFILE) VOL(MUSIC2) OLD
/LOAD PLI
/OPT DECK
COPY: PROC OPTIONS(MAIN);
ON ENDFILE(CONSOLE) GO TO FINISH;
LOOP: GET FILE(CONSOLE) LIST(X);
      PUT FILE(FILE01) LIST(X);
      GO TO LOOP;
FINISH: CLOSE FILE(FILE01);
END COPY;
```

PLITEST

PLITEST is a flexible and efficient tool for examining, monitoring, and controlling program execution. You can use PLITEST interactively or in batch mode.

PLITEST is easy to learn and use. You can begin testing with PLITEST after learning just a few concepts: how to invoke PLITEST, and how to set, display, and remove breakpoints. PLITEST commands are similar to PL/I statements, so you know most of the commands already. And, online help is available at your finger-tips when using PLITEST in an interactive mode.

The IBM publication *OS PL/I Version 2 Programming*: *Using PLITEST*, (SC26-4310-1) describes the how to use PLITEST. It tells you how to select the various testing options that are available, how to set breakpoints, how to inspect and modify variables, and is a complete reference for PLITEST commands. The information below pertains to the specific usage of PLITEST on MUSIC.

On MUSIC only "Line Mode" operation is supported. To use PLITEST you must compile the program with the TEST Compiler Option. You specify this on the /OPT or *PROCESS statement. This option causes the compiler to include the appropriate "hooks" in the generated code to allow subsequent testing of the program. A number of subparameters can be specified on the TEST option that specify what type of testing is to be performed.

The TEST Run-Time Option should also be specified on the /PARM statement when you want to actually test the program. This also allows you to specify a file that contains the PLITEST commands. If none is specified the commands are read from the workstation.

PLITEST keeps a log file of the test session in the file pointed to by the ddname PLILOG. A file statement for this must be included in the test job, even if it is only a dummy file. The log file can be used as input to subsequent test runs. For example, suppose after a long interactive testing session you discover a problem and have to go back to fix it in the source. You can use the commands in the log file to quickly bring you back to where you were before and proceed to debug from that point.

To get online help while in interactive mode, type HELP when prompted for a PLITEST command.

Sample PLITEST session

The following is a listing of the file used in the sample PLITEST session.

```
 /SYS REG=1024
 /FILE PLILOG DUMMY
 /PARM LANGUAGE(EN),TEST
 /LOAD PLI
/OPT SOURCE,TEST(ALL)
  /* PLI TEST PROGRAM */
 TESTP: PROCEDURE OPTIONS(MAIN);
 ON ENDFILE(CONSOLE) GO TO FINISH;

 PUT SKIP LIST ('THIS PROGRAM CALCULATES THE SQUARES OF NUMBERS');

 /* LOOP TO READ INPUT */
 LOOP:  PUT SKIP LIST ('ENTER A NUMBER');
        PUT SKIP;
        GET FILE(CONSOLE) LIST(X);
        Y=X**2;
        PUT SKIP LIST(X,' SQUARED IS ',Y);
        GO TO LOOP;

 /* ALL DONE */
 FINISH: PUT SKIP LIST('ALL DONE');
         END;
```

In the following sample PLITEST session the commands typed in by the user are preceded by the ">" character.

```
>pli.test
 5668-910 IBM OS PL/I OPTIMIZING COMPILER V2.R2.M1    11 DEC 89   12:04
 SOURCE,TEST(ALL);
                    SOURCE LISTING
     STMT
           /*  PL/I TEST PROGRAM  */
        1 TESTP: PROCEDURE OPTIONS(MAIN);
        2 ON ENDFILE(CONSOLE) GO TO FINISH;
        3 PUT SKIP LIST ('THIS PROGRAM CALCULATES THE SQUARES OF NUMBER
          /* LOOP TO READ INPUT */
        4 LOOP:  PUT SKIP LIST ('ENTER A NUMBER');
        5        PUT SKIP;
        6        GET FILE(CONSOLE) LIST(X);
        7        Y=X**2;
        8        PUT SKIP LIST(X,' SQUARED IS ',Y);
        9        GO TO LOOP;
          /* ALL DONE */
       10 FINISH: PUT SKIP LIST('ALL DONE');
       11         END;
 END OF COMPILATION OF TESTP
 00B130 BYTES USED
 EXECUTION BEGINS
 TEST:
>QUERY PROGRAM
 The compile-time data for program TESTP is
     The statement table has the STMT format.
     The program was compiled with the following options:
              TEST(ALL,SYM)
              NOOPTIMIZE
              NOGRAPHIC
              NOINTERRUPT
              CMPAT(V2)
              SYSTEM(MVS)
     This program has no subblocks.
 TEST:
>AT 9
 TEST:
>GO
 THIS PROGRAM CALCULATES THE SQUARES OF NUMBERS
 ENTER A NUMBER
>5
 TEST (TESTP:9):
>LIST(X,Y)
   5.00000E+00
   2.50000E+01
 TEST:
>GO
   5.00000E+00             SQUARED IS              2.50000E+01
 ENTER A NUMBER
>8
 TEST (TESTP:9):
>LIST(X,Y)
   8.00000E+00
```

```
  6.40000E+01
 TEST:
>GO
  8.00000E+00             SQUARED IS              6.40000E+01
 ENTER A NUMBER
>/EOF
 You have been prompted because the ENDFILE ( CONSOLE ) condition
 has been raised in your program.
 The current location is TESTP : 6.
 TEST:
>GO
 You have been prompted because the FINISH condition has been raised in
 has been raised in your program.
 The current location is TESTP : 11.
 TEST:
>GO
 ALL DONE
```

PL/I OS-Mode Loader - PLILG

An input stream that consists entirely of object modules produced by the PL/I Optimizing Compiler will be processed more efficiently if the /LOAD PLILG statement is used to inform the system that this is the case.

Usage - PLILG

This loader is invoked by the use of a /LOAD PLILG statement. It loads and executes a PL/I program in object module form. It is more efficient to use this processor than /LOAD PLI when the input consists solely of object modules. The /JOB control statement may be used with this processor to specify LOADER options.

The statement is used a follows:

```
/FILE statements as required
/LOAD PLILG
...object modules produced by the PL/I Optimizing Compiler
```

Available Unit Numbers and Buffer Spaces - PLILG

The user can use 3 UDS files. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the PL/I transient library modules, which are loaded dynamically during execution.

Parameters - PLILG

```
/JOB  [MAP   ][,NOGO][,LET][,NOPRINT][,NOSEARCH][,FILRFM]
      [FULMAP]
```

Parameters can be separated by one or more commas or blanks.

MAP List a storage map of the loaded program.

FULMAP List an extended storage map of the loaded program.

NOGO Do not load or execute the program.

LET Allow the program to be executed even if unresolved external references (such as missing subroutines) are found.

NOPRINT Informative and diagnostic messages are not to be produced.

NOSEARCH Do not search the subroutine library for unresolved routines.

FILRFM Supplies a RECFM V if the file is V/VC.

Notes:

1. Multiple /JOB statements may be used if desired.

2. The DEBUG, CDUMP, DUMP, IOTRACE and SVCTRACE parameters for OS-mode processors are ignored when used with /LOAD ASMLG, /LOAD COBLG, or /LOAD PLILG. They can be used with /LOAD ASM, /LOAD COBOL, /LOAD PLI, and the other OS-mode processors that do not invoke the OS-mode loader directly.

```
/OPT  [SYSIN=n]
```

SYSIN=n Specifies that input is to be taken from the MUSIC unit number n. A /FILE statement defining MUSIC I/O unit n as the appropriate input file must appear at the beginning of the job. When an end-of-file or a /DATA statement is encountered on this input data set, further input is taken from the normal input stream.

PL/I Load Module Executor - XMPLI

The /LOAD XMPLI statement is used to load and execute a PL/I load module which has previously been stored on a User Data Set (UDS) file or file by the MUSIC Linkage Editor. This is similar to /LOAD XMON. It results in faster loading (compared with /LOAD PLILG) and allows overlay structures to be used.

Usage - XMPLI

This executor is invoked by the use of a /LOAD XMPLI statement. Usage is the same as for /LOAD XMON except that the load module data set must not be on I/O unit number 4.

Available Unit Numbers and Buffer Space - XMPLI

The user can use a maximum of 3 UDS files, one of which may be the data set containing the PL/I load module. MUSIC I/O unit 4 cannot be defined in /FILE statements since it is used for the PL/I transient library.

Restructured Extended Executor - REXX

REXX (Restructured Extended Executor) is a high level language which allows the execution of MUSIC commands and programs from directly within the REXX program itself, and thus, enables the user to chain together sequences of MUSIC commands under program control. The language constructs offered by REXX lend themselves well to structured programming and the powerful functions for command parsing, character handling, and data conversion, make it an excellent tool in the interactive environment. The language is described in the IBM publications *VM/SP System Product Interpreter User's Guide* (SC24-5238) and *VM/SP System Product Interpreter Reference* (SC24-5239). The sections of these publications that describe the usage of REXX in the CMS environment do not generally apply under MUSIC.

MUSIC has a tutorial for learning REXX. Type "TUT" in *Go mode or select the "Tutorials" item for the FSI main menu.

The following describes the usage of REXX in the MUSIC environment.

Invoking REXX

The file containing a REXX program should contain the following control statements.

```
/INCLUDE REXX
  . . . .
  REXX program statements
  . . . .
```

The program can be executed from MUSIC command mode by typing the name of the file containing the program followed by an optional parameter string. For example, if a REXX program was contained in the file REX1, typing:

```
REX1 ABCD
```

would execute the program. The parameter string ABCD would be available to the program via the PARSE ARG instruction or the *ARG* function.

If you find it necessary to use a larger region size for a Rexx program, you must create a private file called REXX, containing:

```
/SYS NOPRINT,REGION=nnnn
/LOAD REXX
```

For example, a larger region would be needed for FSI file management if a very large number of files is involved. The default (and minimum) region size for /LOAD REXX is 1024K.

Executing MUSIC Commands - REXX

MUSIC commands can be executed by simply stating them as character strings, or expressions that result in character strings, at the appropriate location in the program. When the command is completed, control is returned to the REXX program at the statement after the one invoking the command. The variable *RC* is set to the value of the return code as set by the execution of the command.

Examples:

1. Using a character string.

```
'EDIT FILE1'
```

The Editor will be invoked to edit the file called FILE1. When the edit session is finished, control is returned to REXX.

2. Using an expression.

```
name='FILE1'
cmd='EDIT'
cmd name
```

when the line *cmd name* is evaluated, the result will be the character string 'EDIT FILE1' which is passed to MUSIC as a command. This gives the same result as example 1.

Note: Commands that are processed directly by MUSIC's workstation command scanner are currently not supported by REXX. These are /COMPRESS, /CTL, /DISCON, /EXEC, /NS, /PAUSE, /PROMPT, /REQUEST, /RUN, /STATUS, /TIME, /TEXT, /TABIN, /TABOUT, /USERS, /WINDOW.

Communicating with the workstation - REXX

The SAY instruction is used to write data to the workstation. If the first character of the output string is a valid workstation carriage control character, it is treated as such. Input from the workstation can be requested via the PULL or the PARSE EXTERNAL instruction. PULL (or PARSE PULL) first looks for data from the STACK. If this is empty, the data is read from the workstation. (The STACK is described later on.) The PARSE EXTERNAL instruction always reads from the workstation. The following complete REXX program illustrates the use of SAY and PULL.

```
/INC REXX

  /* program to ask the user's name */
  /* and says hello                 */

    say ' Hi there, what is your name.'
    pull name
    say ' hello ' name
```

Accessing the STACK - REXX

REXX maintains an internal stack which is accessed by the PUSH, QUEUE and PULL instructions. All REXX programs in a particular program chain have access to the same stack so data can be passed from one REXX program to another via the stack. It should also be noted that since any REXX program in the chain has access to the stack, they can also change it. This does not present a problem when all the programs in the chain have been designed to work together, but the execution of a command may invoke some REXX program, not planned for in the original design, which modifies the stack, causing errors when it returns. Thus, if a REXX program is to be callable from any other REXX program, care must be taken to preserve the contents of the stack. The MUSIO command can be used to transfer the contents of all or part of the stack to or from the MUSIC Save Library. The following summarizes commands and instructions for accessing the stack.

PULL - get item from top of stack.
PUSH - put an item at top of stack.
QUEUE - put an item at bottom of stack.

QUEUED() - function: returns number of items in stack.

MUSIO - command: transfers stack to and from MUSIC's Save Library.

MUSIO Command - REXX

The MUSIO command provides the interface between REXX and the MUSIC file system. A specified number of records are transferred from the Save Library to the stack or vice versa. The variable RC is set to the MUSIC file system return code. The format of the MUSIO command is as follows....

```
MUSIO option filename num
```

option READ, WRITE, APPEND, CLOSE.

- READ — If required, the file is opened. The requested number of records are read from the file and placed at the end of the stack. If the records are longer than 255 bytes, they are truncated. Subsequent reads will continue at the next record in the file. If an end of file is encountered, RC is set to 1. In this case, the QUEUED function can be used to determine the number of records read. If an error occurs in either the open or reading of the file, the return code (RC) will be non-zero. Since the stack is kept in main storage, reading a large file in one MUSIO request could cause the REXX program to run out of storage.
- WRITE — If required, the file is opened. If the file does not exist, a new file is created. The requested number of records are PULLed from the stack and written to the file. Subsequent writes will continue after the last record written. If the record in the stack is longer than the record length of the file, the record is truncated. If errors occur opening or writing to the file, the return code (RC) will be non-zero.
- APPEND — Similar to WRITE except it causes the file to be opened for append only access.
- CLOSE — The file is closed. This can also be used to rewind a file for subsequent processing.

filename The name of the file to access.

num The number of records to transfer. The string ALL indicates that all records should be transferred. The default is 1.

The following program, which lists a file, shows the use of MUSIO, PULL, and the QUEUED function.

```
/INC REXX

/*   list file XXXXX at the workstation */

    'MUSIO READ XXXXX ALL'  /* read the file */
    nrec=queued()           /* get number of records read */
    do i = 1 to nrec        /* loop to print stack */
      pull rec
      say rec
    end
```

EXEC Command - REXX

Not all programs on MUSIC are executed by one-line commands. More typically, a small file is created containing control statements such as /FILE, /LOAD, and /INCLUDE. This can easily be accomplished from a REXX program. The appropriate control statements are QUEUED on the stack, MUSIO writes the stack to a file, and the file is executed by specifying the file name as a command.

```
/INC REXX

   /* put job in file X1 and execute it */

   queue '/FILE 10 N(PROG.OBJ) NEW(REPL)'
   queue '/LOAD VSFORT'
   queue '/OPT DECK'
   queue '/INC PROG.S'
   'MUSIO WRITE X1 ALL'
   'X1'         /* execute program in X1 */
```

When writing REXX programs to be used by others, the programmer cannot simply choose an arbitrary file name such as X1 to contain the program, since the user might actually have a file called X1. In order to avoid this problem the EXEC command is provided to execute whatever program is in the stack. The stack is written to the the reserved file @EXEC.tcb (tcb- terminal id number) and the program in this file is then executed.

```
/INC REXX

   /* use EXEC command to execute a program from the stack */

   queue '/FILE 10 N(PROG.OBJ) NEW(REPL)'
   queue '/LOAD VSFORT'
   queue '/OPT DECK'
   queue '/INC PROG.S'
   'EXEC'       /* execute what is in the stack */
```

REXLIB Command - REXX

The REXLIB command provides a direct interface to the MUSIC LIBRARY command, avoiding the overhead of scheduling LIBRARY as a separate job. In addition the output from REXLIB can go directly into the REXX stack. Output is never written to the workstation. The parameters are the same as for the standard LIBRARY command with the addition of the "Q" parameter which indicates that any output should be queued in the stack. If you want REXLIB to operate like the DIR command rather than the LIBRARY

command, use the DIR option, for example:

```
REXLIB * Q DIR
```

REXLIB sets the following return codes.

```
  0 - no errors
  1 - not enough memory
  4 - invalid file name specification
  8 - invalid character in file name
 12 - error in user id
 16 - not authorized to look at this library
 20 - invalid character in user id
 24 - invalid parameter
 28 - conflicting parameters
 32 - parameter specified twice
 36 - invalid filename in save parameter
 40 - unable to open file for save
 44 - error accessing index
 48 - error scanning index
 52 - not enough memory to sort the file names
 56 - not enough memory to send output to the REXX stack
900 - memory work area is too small
```

The following example displays a list of a users files.

```
/INC REXX

   'REXLIB * Q'
   nfiles=queued()
   say 'You have the following files'
   do i = 1 to nfiles
      pull filename
      say filename
   end
```

Program Chaining - REXX

When REXX is first invoked, it establishes itself as an *always* program. This means that it will always be given back control on the termination of any program or command it schedules. It is this technique of allowing program and command chaining that makes REXX so powerful. However, this chain can be broken by a /CAN ALL command or if one of the scheduled programs replaces REXX as the *always* program. The REXX *always* program is contained in the file called REXX. By default there is a public file on the system called REXX, which contains the default control statements defining the region size an time limits. If these defaults are not sufficient, they can be overridden by creating a private file called REXX with the appropriate parameters on the /SYS statement. Note that the /SYS statement for the initial REXX program is NOT carried over between scheduled commands. In addition since all REXX programs that are chained together should use the SAME region size, it is recommended that anyone writing a REXX program for general use, start the program with /INC REXX (not /LOAD REXX), so that the /SYS statement from the user's or system REXX file is used throughout.

Interface with the PANEL subsystem - REXX

An interface to the PANEL facility is available through a special PANEL command in REXX. The screens are created as usual via the PANEL facility and stored as object files in the Save Library. When REXX encounters a PANEL command it dynamically loads the appropriate object files for the screen images. Modifiable fields are then filled from the specified REXX variables and the panel service routine is invoked to perform the I/O. When this is complete, the data from any modified fields is then returned via the REXX variables.

For more information on this interface, refer to *Chapter 10. Utilities* under PANEL.

Callable MUSIC System Functions - REXX

The following MUSIC system subroutines are callable from REXX. Each parameter should be quoted, otherwise unexpected results may occur. For example,

```
CALL TSUSER '2','USERID'
```

returns the user id in variable 'USERID'.

NXTCMD	CLRIN	GETRET
TSTIME	NOECHO	EOJ
TSUSER	ECHOIN	PARM
FILMSG	NOSHOW	
DELAY	NOTRIN	
KEEPIN	TRIN	

Editor Macro Facility - REXX

Editor macros can be written in the REXX language. The macros can issue Editor commands, extract information from the Editor using the EXTRACT command, call other macros, and use the MUSIO and PANEL commands. The macros cannot issue MUSIC commands.

For more information on this facility, refer the topic "Editor Macro Facility in REXX" in *Chapter 7. Using the Editor*.

Sample REXX programs

Example 1:

This program prompts the user for commands and executes them.

```
/INC REXX
/* ask for the user's name */
say " Welcome to REXX, what's your name?"
pull name
say ' ' name ', when asked to enter a command, type in a MUSIC'
say " command of your choice and I'll see it gets done. If you"
say ' have trouble with the syntax of a particular command just'
say ' enter "HELP cmd" where cmd is the command name.'

  /* main prompt loop */

   do forever
     say ' Go ahead' name
     pull cmd   /*get command*/
     cmd        /*execute it */
   end
```

Example 2:

Calculate factorial. Note the recursive nature of the REXX language.

```
/INC REXX
/* calculate factorial */
parse arg num rest
if datatype(num,'w') •= 1 then do
    say 'I can only calculate the factorial of whole numbers.'
    exit
end
say 'The factorial of' num 'is' fact(num)
exit

fact: procedure
   arg num
   if num = 1 then return 1
   temp = fact(num-1)
   return num * temp
```

RPG II - RPG

OS RPG II (Program Product 5740-RG1) is an enhanced version of RPG, providing users with an efficient technique for developing programs to:

1. obtain data records from input files
2. perform calculations
3. write reports
4. use table look up
5. maintain files
6. generate complete RPG II source programs from simplified specifications, and standard RPG II source specifications
7. sort input records

Control Statements - RPG

A complete list of the RPG II and the Auto Report control statements is available in the two guides listed below.

Usage - RPG

The following is a description of the /LOAD statements for RPG II on MUSIC:

Statement	Description
/LOAD RPGAUTO	Autoreport execution and resulting RPG II program compilation and execution.
/LOAD RPG	RPG II source program compilation and execution.
/LOAD RPGLG	Execution of an RPG II program from object modules produced by the RPG II compiler.
/LOAD XMRPG	Execution of an RPG II program from load modules produced by the MUSIC linkage editor. (Similar to /LOAD XMON.)

Example:

```
/FILE ... (if required)
/LOAD RPGAUTO
/JOB ...  (loader options, if required)
/OPT ...  (compiler options, if required)
.
.         (Autoreport source program)
.
/DATA
.
.         (data, if required)
.
```

The following options are available for the /OPT statement in RPGAUTO or RPG jobs:

Option	Description
LIST	Produces an Autoreport and/or compiler listing on SYSPRINT (default).
NOLIST	Suppresses the list option.
DECK	Produces an object module from the compiled program on SYSLIN (default).
NODECK	Suppresses the DECK option.

The following is a list of DDNAMES used by the various steps. These names are automatically provided by MUSIC, so a /FILE statement is not required for them. However, an overriding /FILE statement may be used if desired (for example, to direct SYSLIN output to a file).

Step	ddname	Description
RPGAUTO	SYSIN	Source program input.
	SYSPRINT	Program listing output.
	SYSPUNCH	Output for the RPG II compiler.
	SYSLIN	Output object code if SORT/SELECT is specified.
	SYSLIB	Copy library containing the source code referenced by the Autoreport /COPY statement.
RPG	SYSIN	Source program input.
	SYSPRINT	Program listing output.
	SYSLIN	Object code output.
	SYSUT1	Compiler work file.
	SYSUT2	Compiler work file.

Other DDNAMES used by the program must be defined by separate /FILE statements, placed before any /LOAD statement.

If RPG printed output (ie. output to a PRINTER device) is to be directed to a MUSIC file, a /FILE statement must be included in the job specifying a record length of 133 (LRECL(133)) and a record format of either fixed or fixed compressed (RECFM(F) or RECFM(FC)). For example:

```
/FILE REPORT NAME(REPORT.FILE) LRECL(133) RECFM(F) ...
```

Keyed or indexed files, using VSAM, are supported. Direct access by the chain and record-address file techniques, is not supported on MUSIC.

The /COPY Statement of Autoreport - RPG

The '/COPY' statement of Autoreport is supported on MUSIC by the use of a PDS (partitioned data set), whereby members of this PDS are actually individual files in the user's library or listed in the common file index. A /FILE statement with the DDNAME SYSLIB specifying PDS name '*.RPGCOPY' is automatically supplied by MUSIC. In this way, any name specified by an Autoreport /COPY statement will have the suffix '.RPGCOPY' added to it and the system will search the appropriate Save Library for the resulting file. For example, with the /COPY statement:

```
/COPY R,ABC
```

the MUSIC file ABC.RPGCOPY will be copied into the Autoreport program.

Sample Programs - RPG

The following two RPG II programs are provided with the system. They are:

RPG.TEST1 - RPG II program using Autoreport.

RPG.TEST2 - RPG II program.

Reference Manuals - RPG

DOS/VS RPG II Auto Report, IBM Form No. SC33-6034.

OS/VS RPG II Addendum to DOS/VS RPG II, Auto Report, IBM Form No. SC33-6128.

DOS/VS RPG II Language, IBM Form No. SC33-6031.

OS/VS RPG II Addendum to DOS/VS RPG II Language, IBM Form No. SC33-6129.

DOS/VS RPG II Messages, IBM Form No. SC33-6033.

OS/VS RPG II Addendum to DOS/VS RPG II Messages, IBM No. SC33-6130.

SAA RPG/370

SAA RPG/370 Program Product 5688-127, implements the high-level programming language defined by the SAA RPG Common Programming Interface (CPI). SAA RPG/370 may be used for writing a full range of batch and interactive programs that:

1. obtain data records from input files
2. perform calculations
3. write reports
4. use table look up
5. maintain files
6. generate complete SAA RPG/370 source programs from simplified specifications, and standard SAA RPG/370 source specifications
7. sort input records

Control Statements - RPG370

A complete list of the SAA RPG/370 control statements is available in the two guides listed below.

Usage - RPG370

The following is a description of the /LOAD statements for SAA RPG/370 on MUSIC:

/LOAD RPG370 SAA RPG/370 source program compilation and execution.

/LOAD XMRPG370 Execution of an SAA RPG/370 program from load modules produced by the MUSIC linkage editor. (Similar to /LOAD XMON.)

Example - RPG370

```
/FILE ... (if required)
/LOAD RPG370
/JOB ...  (loader options, if required)
/OPT ...  (compiler options, if required)
.
.         (RPG/370 source program)
.
```

The following options are available for the /OPT statement:

SOURCE Produces a compiler listing on SYSCPRT. (Default on batch).

NOSOURCE Suppresses the compiler listing option. (Default on a workstation).

TEXT Produces an object module from the compiled program on SYSLIN (default).

NOTEXT Suppresses the TEXT option.

LIST Produce an pseudo assembler listing of the program.

NOLIST No pseudo assembler listing created (default).

SAAFLAG Specifies whether the compiler flags specifications not supported by SAA RPG.

NOSAAFLAG No SAA RPG specification checking is performed (default).

XREF A cross-reference list for the source program is produced.

NOXREF No cross-reference is produced (default).

The following is a list of DDNAMES used by the various steps. These names are automatically provided by MUSIC, so a /FILE statement is not required for them. However, an overriding /FILE statement may be used if desired (for example, to direct SYSLIN output to a file).

Step	DDNAME	LRECL	Description
RPG370	SYSIN	80	Source program input.
	SYSCPRT	133	Program listing output.
	SYSLIN	80	Object code output.
	SYSUT1	80	Compiler work file.
	SYSUT4	80	Compiler work file.
	SYSUT7	3200	Compiler work file.
	SYSUT8	3200	Compiler work file.
	SYSUT9	80	Compiler work file.
	SYSIT	4096	Compiler work file.
	SYSTERM	133	Compiler error messages
	SYSPRINT	133	Compiler work file.
	DUMPRPG	133	Formatted RPG dump file
	SYSMSG	150	Compiler message files
	SYSMSG1	3200	Compiler/Runtime message file
	SYSMSG2	3200	Compiler/Runtime message file

Other DDNAMES used by the program must be defined by separate /FILE statements, placed before any /LOAD statement.

When SAA RPG output to a PRINTER device, the following options MUST be specified on the /FILE statement. The options are:

```
OSRECFM(FBA) OSLRECL(xxx)  OSBLK(xxx)
```

where xxx is the length of the output line, normally 132.

For example, to have the output display at the workstation specify:

```
/FILE REPORT PRT OSRECFM(FBA) OSLRECL(132)
```

To direct the output to a file, specify:

```
/FILE REPORT NAME(REPORT.OUT) NEW(REPL) LR(132)
/ETC   OSRECFM(FBA) OSLRECL(132)  OSBLK(132)
```

Keyed or indexed files, using VSAM, are supported.

References - RPG370

SAA RPG/370 Programming Guide, Form No. SC09-1335. *SAA RPG Reference Summary*, Form No. SX09-1164. *SAA RPG/370 General Information*, Form No. GC09-1336. *SAA RPG Reference*, Form No. SC09-1286.