# Using FORTRAN at the NIH Computer Center

September 1998

Center for Information Technology

National Institutes of Health
Center for Information Technology
NIH Computer Center
12 South Drive MSC 5607
Bethesda, Maryland 20892-5607

Publication No. CIT183

## Table of Contents

# 1 INTRODUCTION

This manual describes the use of the FORTRAN programming language at the NIH Computer Center. This manual is intended to give programmers the FORTRAN information they need in order to create new programs and to maintain programs running on the MVS South System. The information in this manual should be used in conjunction with the *NIH Computer Center User's Guide*, *Batch Processing and Utilities at NIH*, and the manuals described in Section 3 of this publication.

The FORTRAN programming language receives full (Level 1) support. Questions on FORTRAN should be directed to the Technical Assistance and Support Center (TASC), either by phone at (301) 594-3278 or by submitting a Problem Tracking Report (PTR). There are several methods of submitting a PTR:

- World Wide Web
  Users with NIHnet or Internet connections can submit a PTR through the World Wide Web. To access the PTR system, connect to:

  http://datacenter.cit.nih.gov/ptr.html

- Electronic Mail
  PTRs can also be submitted to the Computer Center by sending electronic mail to the WYLBUR initials PTR or the Internet address PTR@CU.NIH.GOV. Mailed PTRs must have a valid SUBJECT header containing the submitter's name and telephone number, and be of the form

  Subject: PTR FROM name TELEPHONE phone-number

  For example:

  Subject: PTR FROM Tom Jones TELEPHONE 6-1111

- ENTER PTR
  Users can submit a PTR is through WYLBUR's ENTER PTR command.

Changes that affect the use of the FORTRAN language will be fully tested and pre-announced through the *Interface* newsletter. For a full description of Level 1 support, see the *NIH Computer Center User's Guide*.

FORTRAN is a programming language that receives widespread use in scientific, engineering, and biomedical applications that involve mathematical computations. The IBM VS FORTRAN Version 2 compiler is used at the NIH Computer Center.

There is a Federal Information Processing Standard (FIPS) for this language. Certain special features and extensions of the language fall outside the FIPS standard. Programs written using only FIPS approved features can be transported more readily between federal installations and different vendors' mainframes. Federal policy encourages the use of features within FIPS standards. For FORTRAN, the ANSI standard has been adopted as the FIPS standard. FORTRAN statements not conforming to the ANSI X3.9-1978 standard are clearly identified in the *VS FORTRAN Version 2 Language and Library Reference*, SC26-4221. In addition, the compiler option FIPS(F) may be specified to flag FORTRAN statements not defined in the current standard.

## 1.1 Procedure Names

Note: the Binder now performs the link-editing functions previously performed by the Linkage Editor.

The procedure names used in this manual are:

FORVCOMP
FORVOBJ
FORVLKGO
FORVLDGO
FORVLKMM
FORVLKSM
FORVCALL

Each procedure name follows the pattern:

lllvffff

| | | |
|---|---|---|
| where | "lll" | is the language prefix (FOR for FORTRAN) |
| | "v" | is the version (V for VS FORTRAN) |
| | "ffff" | is the function |

The meaning of each function is given below:

| | |
|---|---|
| COMP | compilation only |
| OBJ | compile and store object module |
| LKGO | use the Binder (formerly the Linkage Editor) and execute program |
| LDGO | use the Loader and execute program |
| LKMM | use the Binder to store a link-edited load module into an existing |

|  |  |
|---|---|
|  | multi-member PDS |
| LKSM | use the Binder to store a link-edited load module into a new single-member PDS |
| CALL | execute a fully link-edited load module |

In the examples throughout this manual, the following conventions apply:

|  |  |
|---|---|
| "aaaa" | the account number |
| "iii" | the programmer's registered initials |
| "dsname" | name of data set |
| "progname" | name of program stored in partitioned data set (PDS) |
| "fileser" | volume serial number of disk where data set is located; required only if the data set is not cataloged |
| "primary" | primary quantity requested in the SPACE parameter |
| "blocks" | number of directory blocks |
| "stepname" | name of step which executes the procedure; should be unique within a job |
| "ddname" | user-supplied ddname; should be unique within job step |

# 2 DCB INFORMATION FOR FORTRAN DATA SETS

These sections list DCB (Data Control Block) characteristics for FORTRAN data sets.

## 2.1 DCB Information for SYSOUT Data Sets

Listed below are the default record formats and blocksizes for all SYSOUT data sets in the VS FORTRAN procedures:

| PROCEDURE NAME | STEP NAME | DD NAME | DEFAULT RECFM/BLKSIZE | |
|---|---|---|---|---|
| FORVCOMP | COMP | SYSTERM | VA | 244 |
| | | SYSPRINT | VBA | 141 |
| | | SYSUDUMP | VBA | 1632 |
| FORVLKGO | LOAD | SYSPRINT | FA | 121 |
| | | SYSUDUMP | VBA | 1632 |
| | GO | FT06F001 | UA | 133 |
| | | FT07F001 | F | 80 |
| | | FT15F001 | UA | 133 |
| | | FT16F001 | F | 80 |
| FORVOBJ | COMP | SYSTERM | VA | 244 |
| | | SYSPRINT | VBA | 141 |
| | | SYSUDUMP | VBA | 1632 |
| FORVLKSM | LOAD | SYSPRINT | FA | 121 |
| | | SYSUDUMP | VBA | 1632 |
| FORVLKMM | LOAD | SYSPRINT | FA | 121 |
| | | SYSUDUMP | VBA | 1632 |
| FORVCALL | GO | FT06F001 | UA | 133 |
| | | FT07F001 | F | 80 |
| | | FT15F001 | UA | 133 |
| | | FT16F001 | F | 80 |
| FORVLDGO | GO | SYSLOUT | FBSA | 121 |
| | | FT06F001 | UA | 133 |
| | | FT07F001 | F | 80 |
| | | FT15F001 | UA | 133 |
| | | FT16F001 | F | 80 |

**Figure 1. SYSOUT DCB Information for VS FORTRAN Procedures**

## 2.2 DCB Information for Other FORTRAN Data Sets

VS FORTRAN supplies default DCB characteristics for all data sets it accesses, i.e. FT00F001-FT99F001. Besides the SYSOUT data sets whose DCBs are specifically provided by the VS FORTRAN procedures, the remaining data sets will use the FORTRAN-supplied defaults unless overridden by the user’s own JCL. The defaults are given in the following figure.

| **DDNAME** | **SEQUENTIAL DATA SETS** | | | | **DIRECT ACCESS DATA SETS** | | |
|---|---|---|---|---|---|---|---|
| | **RECFM*** | **LRECL**** | **BLK-SIZE** | **BUFNO** | **RECFM** | **LRECL or BLKSIZE** | **BUFNO** |
| FT05F001 and FT07F001 | F | 80 | 80 | 2 | F | The value specified as the maximum size of a record in the OPEN statement | 2 |
| FT06F001 | UA | 133 | 133 | 2 | F | | 2 |
| ALL OTHERS | U | | 800 | 2 | F | | 2 |

*   For records not under FORMAT control, the default is VS.
** For records not under FORMAT control, the default is 4 less than shown.

**Figure 2. DCB Information for FORTRAN Data Sets**

# 3 FORTRAN PUBLICATIONS

The CIT Technical Information Office distributes general information, technical and vendor publications and certain software to the user community. *Using FORTRAN at the NIH Computer Center* is one of the many publications available online through the World Wide Web at:

> http://datacenter.cit.nih.gov/cfb.pub.txt.html

Users may order publications in the following ways:

- Using the World Wide Web, visit:

  http://livewire.nih.gov/publications/publications.asp

  and select the option for ordering publications online. Some publications may not be available through this ordering system.

- Sign on to WYLBUR and use the ENTER PUBWARE command to order publications.
- If you cannot order a publication online, you may place an order by visiting TASC in Building 12A or by telephone.

The following manuals relevant to VS FORTRAN can be ordered:

*VS FORTRAN Version 2 Programming Guide for CMS & MVS*, SC26-4222

> This publication provides information on how to design, debug, test, and execute FORTRAN Version 2 programs. It is not intended as a reference manual.

*VS FORTRAN Version 2: Language and Library Reference*, SC26-4221

> This publication describes each syntactic element available in FORTRAN 77 and the types of subprograms in the VS FORTRAN Version 2 Library. It also contains library, execution-time, and operator messages. There is information about the method used in the library to compute a mathematical function and the effect of an argument error on the accuracy of the answer returned.

*Interface*

> This is a series of technical notes for users, published by the Computer Center. All changes to Computer Center standards and facilities are announced in this publication.

## 4 FORTRAN COMPILER OPTIONS

The default options specified when using Computer Center cataloged procedures for the VS FORTRAN compiler are listed below.

| | |
|---|---|
| CHARLEN(500) | Specifies the maximum length of any CHARACTER variable, CHARACTER array element, or CHARACTER function. |
| FLAG(I) | All levels of diagnostic messages, including information, will be written. |
| FIXED | The input source program is in fixed format. |
| GOSTMT | Internal sequence numbers will be generated for a calling sequence to a subprogram. |
| LANGLVL(77) | The input source program is written in the current FORTRAN language. |
| LINECOUNT(60) | Number of lines per page of compiler listing. |
| NAME(MAIN) | The main program name is MAIN; available only with LANGLVL(66). |
| MAP | Produces table of source program names and statement labels. |
| OBJECT | Object module will be produced. |
| SDUMP(ISN) | Symbolic dump information is available. Execution time option ABSDUMP must be specified to generate the output. |
| SOURCE | Source listing will be produced. |
| SRCFLG | Error messages will be inserted in the source listing. |
| TERMINAL | Error messages and compiler diagnostics will be written on the output data set; a summary of error messages will be printed. |
| XREF | Produces a cross reference listing. |

Additional compile-time options generated are:

AUTODBL(NONE)
CI(no default number)
DC(no default name)
IL(DIM)
NOFIPS
NOIL
NOLIST
NOOPTIMIZE.
NORENT
NOSXM
NOSYM
NOTEST
NOTRMFLG

The standard batch execution-time options generated for the VS FORTRAN compiler are:

NOABSDUMP
NOAUTOTASK
SPIE
STAE
XUFLOW.

For detailed explanations of these options, see chapter 10 of the *VS FORTRAN Version 2: Programming Guide for CMS & MVS*, SC26-4222.

If any options are to be changed, the new values must appear in the options list on the EXEC statement. The OPTIONS symbolic parameter should be used in place of the PARM parameter. Use of the OPTIONS symbolic parameter is illustrated in the examples later in this manual.

For those who must override or augment the cataloged procedures, the stepnames used in the procedure are given in each section.

# 5 COMPILING AND RUNNING FORTRAN PROGRAMS

The procedures in this section are used to compile, link-edit (using the Binder), and execute FORTRAN programs.

## 5.1 Using the Compiler

The COMP procedure provides the user with a one-step procedure to compile FORTRAN source code for diagnostic messages; and, if compilation is successful, to prepare the input for further processing (e.g., the LKGO procedure). This procedure stores the output of the compiler into a temporary data set to be used later in the job and then deleted.

**Symbolic Parameters for FORVCOMP**

| **Required** | **Value to be supplied** |
|---|---|
| None | None |
| **Optional** | **Value to be supplied** |
| OPTIONS=parms | Compiler parameters |
| CORE=nnnnK | Region for the COMP step; 2500K is the default |
| LIBNAME=‘aaaaiii.dsname’ | Dsname of first user-defined library for compiler input |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME=‘aaaaiii.dsname’ | Dsname of second user-defined library for compiler input |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The internal stepname for the FORVCOMP procedure is COMP.

**Example 1:**

To compile a VS FORTRAN program.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
```

**Example 2:**

To compile a VS FORTRAN program with high-level optimization and produce an assembly listing.

```
//stepname EXEC FORVCOMP,OPTIONS='OPT(3),LIST'
//COMP.SYSIN DD *
  (source program)
```

## 5.2 Using the Binder

Note: the Binder now performs the link-editing functions previously performed by the Linkage Editor.

The LKGO procedure performs the following functions:

- link-edits the program to prepare a load module for execution
- executes the load module

The LKGO procedure provides the user with the DD statements needed to use the printer (FT06F001 or FT15F001). The ddname SYSIN may be used in place of FT05F001. All VS FORTRAN execution-time error messages are written to the DD statement FT06F001; because of a local system standard, this should not be overridden. The user must provide JCL for any additional I/O units (data sets) used. Section 6.4 discusses specifying user-defined libraries with LKGO.

Programs that need large amounts of storage (e.g. for large arrays), can benefit from the use of 31-bit addressing to access the extended address space. See Example 5 below.

There must be one GO.ddname DD statement describing each data set used. DD statements to override ddnames within the procedure must precede those for ddnames to be added to the procedure. See the manual *Batch Processing and Utilities at NIH* for a description of the format of DD statements.

**Symbolic Parameters for FORVLKGO**

| **Required** | **Value to be supplied** |
|---|---|
| None | None |
| **Optional** | **Value to be supplied** |
| OPTIONS=parms | Binder parameters |
| CORE=nnnnK | Region for GO step; 4096K is the default |

| | |
|---|---|
| LIBNAME='aaaaiii.dsname' | Dsname of first user-defined library |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME='aaaaiii.dsname' | Dsname of second user-defined library |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The stepname within the FORVLKGO cataloged procedure is LOAD for the link-edit step and GO for the run step.

**Example 3:**

To compile the VS FORTRAN main program and its subroutines, and execute it. The execution-time parameter ABSDUMP causes all program variables to be printed out if the program ends abnormally.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN  DD  *
   (source program)
//stepname EXEC FORVLKGO,PARM.GO=ABSDUMP
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 4:**

To compile the VS FORTRAN main program and subroutines and execute it. The OPTIONS parameter in the compile step requests an Assembler language listing. The OPTIONS parameter in the run step requests the Binder option XREF. The CORE parameter supplies a larger region size for the GO step.

```
//stepname EXEC FORVCOMP,OPTIONS=LIST
//COMP.SYSIN  DD  *
   (source program)
//stepname EXEC FORVLKGO,CORE=nnnnK,OPTIONS=XREF
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 5:**

To compile, link-edit, and execute an existing program. The OPTIONS parameter in the compile step specifies dynamic common and names the COMMON areas to be located above the line in the 31-bit address area. The OPTIONS parameter in the run step requests the Binder option AMODE=31 to indicate that the load module should use 31-bit addresses and therefore be able to address any storage location (either above or below the line).

```
//stepname  EXEC  FORVCOMP
//    OPTIONS='DC(ABIG1,BBIG1)'
//COMP.SYSIN   DD *
      COMMON /ABIG1/ AARRAY(200,100,50),
     X      BARRAY(500,1000)
      COMMON /BBIG1/ AMATRX(10000,250),BMATRX(300),
     X      CMATRX(50,500)
         .
 (rest of source program)
         .
//stepname EXEC FORVLKGO,OPTIONS='AMODE=31'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
  (data)
```

## 5.3 Creating and Using Object Modules

The OBJ procedure is used to compile source code and store the resultant object module into a sequential data set. The output of this procedure must be processed by the Binder before it can be run. The LKGO procedure may be used to link-edit and execute the object module(s) created by an OBJ procedure.

**Symbolic Parameters for FORVOBJ**

| **Required** | **Value to be supplied** |
|---|---|
| NAME='aaaaiii.dsname' | Dsname of object module to be stored |

| **Optional** | **Value to be supplied** |
|---|---|
| DISK=fileser | Required only for a data set written to a dedicated disk |
| STORAGE=type | Unit name for the object module; FILE is the default |
| OPTIONS=parms | Compiler parameters |
| CORE=nnnnK | Region for the COMP step; 2500K is the default |
| STATUS=status | NEW is the default; use OLD to replace an existing data set |
| SIZE=primary | Primary space allocation for object module; default is 1000 |
| UNITS=type | Allocation units for object module; the default is blocks of 1024 bytes |
| LIBNAME='aaaaiii.dsname' | Dsname of first user-defined library for compiler input |
| LIBDISK=fileser | Volume for first library; required only if the data set |

| | |
|---|---|
| | is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME=‘aaaaiii.dsname’ | Dsname of second user-defined library for compiler input |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The internal stepname for the FORVOBJ procedure is COMP.

**Example 6:**

To compile a VS FORTRAN program and save the object module.

```
//stepname EXEC FORVOBJ,NAME=‘aaaaiii.dsname’
//COMP.SYSIN  DD  *
  (source program)
```

**Example 7:**

To compile a VS FORTRAN program, save the object module into an existing data set. Former contents will be destroyed.

```
//stepname EXEC FORVOBJ,STATUS=OLD,
//  NAME=‘aaaaiii.dsname’
//COMP.SYSIN  DD  *
  (source program)
```

To execute a program which has been stored by an OBJ procedure, use the LKGO procedure. The user must supply a //LOAD.SYSLIN DD statement describing the data set containing the program which was compiled and saved.

**Example 8:**

To compile a VS FORTRAN program, save the object module, link-edit, and run. The object module saved as “aaaaiii.dsname1” from the OBJ procedure is used as input for the link-edit step.

```
//stepname EXEC FORVOBJ,NAME=‘aaaaiii.dsname1’
//COMP.SYSIN  DD  *
   (source program)
//stepname EXEC FORVLKGO
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 9:**

To execute a VS FORTRAN main program and subroutines that have been created as separate data sets by the OBJ procedure. The user must supply a DD statement for each data set that contains a program or subroutine and make sure the main program is defined first.

```
//stepname EXEC FORVLKGO
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,
//    DISP=SHR
//   DD  DSN=aaaaiii.dsname2,DISP=SHR
//   DD  DSN=aaaaiii.dsname3,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 10:**

To execute a VS FORTRAN program where the main program is to be compiled and the subroutines have been stored by the OBJ procedure in two data sets. These data sets will be concatenated with the data set created by the COMP step.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLKGO
//LOAD.SYSLIN DD
// DD DSN=aaaaiii.dsname1,DISP=SHR
// DD DSN=aaaaiii.dsname2,DISP=SHR
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

## 5.4 Using the Loader

The LDGO procedure combines the link-edit and run steps into one. The Loader will accept object modules and load modules. It will also search libraries defined by the SYSLIB DD statement within the procedure if unresolved external references remain after processing the primary input defined by the SYSLIN DD statement within the procedure. It provides the user with the DD statements needed to use the printer (FT06F001 or FT15F001). The ddname SYSIN may be used in place of FT05F001. All FORTRAN execution-time error messages are written to the DD statement FT06F001; because of a local system standard, this should not be overridden. The user must provide JCL for any additional I/O units (data sets) used.

The LDGO procedure should be used during the early stages of program development (debugging); it is particularly recommended for the development of small and medium-sized programs. Using LDGO is often more economical than using LKGO, but a dump from a LDGO run may not be sufficient to resolve a problem. If so, the job may have to be rerun using the Binder (LKGO).

Additional technical information on the use of the Loader is given in the manual *Batch Processing and Utilities at NIH*.

**Symbolic Parameters for FORVLDGO**

| **Required** | **Value to be supplied** |
|---|---|
| None | None |

| **Optional** | **Value to be supplied** |
|---|---|
| OPTIONS=parms | Loader and GO parameters |
| CORE=nnnnK | Region for GO step; 1000K is the default |
| EPT=entry | Entry point for main program; the default is MAIN |
| LIBNAME=‘aaaaiii.dsname’ | Dsname of first user-defined library |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME=‘aaaaiii.dsname’ | Dsname of second user-defined library |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The internal stepname for the FORVLDGO procedure is GO.

**Example 11:**

To compile a VS FORTRAN main program and use the Loader to execute it.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN  DD  *
   (source program)
//stepname EXEC FORVLDGO
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

# 6 STORING AND USING PROGRAMS IN USER LIBRARIES

The following procedures were developed to store user programs in load module form. Users can develop and maintain their own private libraries, which are partitioned data sets.

A Partitioned Data Set (PDS) is divided into one or more sequential “members”, each of which may be accessed independently. Each member has a unique name, up to 8 characters long, stored in a directory. The directory contains an entry for each member consisting of the member name and a pointer to the location of the member in the data set. When a member is deleted or replaced, only the member-name-pointer is deleted or changed. The space used by the member cannot be reused until the data set is condensed. If there is not enough space for a new or replacement member, or if there are no more free entries in the directory, no members can be added. A job that attempts to add a new member to a PDS that is full usually ABENDs with an X37 completion code. A PDS must be stored on a disk and cannot exceed one disk pack in size.

Load modules (the output from the Binder) must be stored in PDSs. The programs may be either fully or partially link-edited. The Binder will automatically search libraries defined by the SYSLIB DD statement to resolve calls or references to programs that are not included in the main input stream defined by the SYSLIN DD statement. The libraries are searched in the order they are defined. When a reference is found, no further searching is done, and the next search begins again at the first library. If all external references and subroutine calls are resolved, the program is fully link-edited and is, therefore, directly executable without link-editing again. If the external references and calls are not to be resolved, the NCAL option must be specified on the EXEC statement for the procedure used to store the program. The program is then partially link-edited and must be reprocessed by the Binder before it can be executed.

Executing fully resolved load modules may cost less because a link-edit step is saved every time the program is run; however, problems may develop as a result of updates to the computer system. Fully resolved load modules cannot take advantage of some of these system improvements. In addition, a program may fail to run if it contains old interfaces to system modules.

To avoid these problems, fully resolved load modules should be re-created periodically, particularly whenever a new system release is installed. If re-creating the fully resolved modules is difficult, it may be better to keep partially resolved modules and do the final link-edit each time the program is run.

## 6.1 Storing Programs in Single-Member User Libraries

The LKSM procedure is used to link-edit and store a load module (output of the Binder) into a single-member partitioned data set (PDS). The COMP and OBJ procedures may be used to prepare input for the LKSM procedure. A short step, executed before the link-edit step, deletes the PDS if it already exists. Then the link-edit step creates the new data set. If the data set does not already exist, the delete step issues a message, but does not affect later processing.

The user may define two private call libraries for resolving external references. They are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the FORTRAN language libraries and before NIH.UTILITY.

**Symbolic Parameters for FORVLKSM**

| **Required** | **Value to be supplied** |
|---|---|
| NAME='aaaaiii.dsname' | Dsname of PDS to receive load module |

| **Optional** | **Value to be supplied** |
|---|---|
| DISK=fileser | Volume for PDS; required only if the data set is not cataloged |
| STORAGE=type | Unit name for PDS; FILE is the default |
| OPTIONS=parms | Binder parameters |
| PROGRAM=progname | Member name for load module; the default is MAIN |
| SIZE=primary | Primary space allocation for load module; the default is 100 units |
| UNITS=type | Allocation units for load module; the default is blocks of 1024 bytes |
| INCR=secondary | Number of units in each secondary allocation; the default is 12 |
| STEPEND=disp | Disposition for the load module; the default is KEEP |
| UNUSED= | Nullifying causes retention of unused space; the default is RLSE |
| INDEX=blocks | Number of directory blocks for load module PDS; the default is 1 |
| LIBNAME='aaaaiii.dsname' | Dsname of first user-defined library |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME='aaaaiii.dsname' | Dsname of second user-defined library |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The internal stepnames for the FORVLKSM procedures are SCRATCH, for the step to scratch the data set if it already exists, and LOAD for the link-edit step.

**Example 12:**

To compile, fully link-edit, and store a VS FORTRAN program into a single-member PDS.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLKSM,NAME='aaaaiii.dsname'
```

**Example 13:**

To compile, fully link-edit, and store a VS FORTRAN program, overriding the default space allocation. If the program requires more than the default space allocation, the SIZE parameter should be used. The default SIZE allows the user to obtain at least 10 tracks (unnecessary space is released) for the load module.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLKSM,NAME='aaaaiii.dsname',
//  SIZE=primary
```

**Example 14:**

To fully link-edit and store a VS FORTRAN main program using input from the OBJ procedure. The user must supply a DD statement for each data set that contains a program or subroutine and insure that the main program is defined first.

```
//stepname EXEC FORVLKSM,NAME='aaaaiii.dsname'
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,DISP=SHR
// DD DSN=aaaaiii.dsname2,DISP=SHR
// DD DSN=aaaaiii.dsname3,DISP=SHR
```

**Example 15:**

To compile a VS FORTRAN main program (FORVCOMP), link-edit using subroutines previously compiled with the OBJ procedure, and create a fully resolved single-member load module (FORVLKSM).

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLKSM,NAME='aaaaiii.dsname'
//LOAD.SYSLIN DD
//  DD  DSN=aaaaiii.dsname1,DISP=SHR
//  DD  DSN=aaaaiii.dsname2,DISP=SHR
```

## 6.2 Storing Programs in Multi-Member User Libraries

The procedures described below enable the user to add programs to multi-member partitioned data sets and execute them. Before using these procedures, see the manual *Batch Processing and Utilities at NIH* for information on how to establish and maintain partitioned data sets. These procedures differ from the OBJ and LKSM procedures in that many programs can be stored in one data set. The OBJ and LKSM procedures store only one program in one data set.

The LKMM procedure adds a program to a private partitioned data set. If the program name already exists in the data set, it will be replaced. The Binder input is the same as for the LKGO procedure.

The user may define two private call libraries for resolving external references. They are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the FORTRAN language and before NIH.UTILITY. If no libraries are to be searched (no external references are to be resolved), OPTIONS=NCAL must be specified for the LKMM step; this creates a partially link-edited load module.

**Symbolic Parameters for FORVLKMM**

| **Required** | **Value to be supplied** |
|---|---|
| NAME='aaaaiii.dsname' | Dsname of PDS to receive load module |
| PROGRAM=progname | Program name; member name in PDS |

| **Optional** | **Value to be supplied** |
|---|---|
| DISK=fileser | Volume for PDS; required only if the data set is not cataloged |
| STORAGE=type | Unit name for PDS; FILE is the default |
| OPTIONS=parms | Binder parameters |
| LIBNAME='aaaaiii. dsname' | Dsname of first user-defined library |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME='aaaaiii.dsname' | Dsname of second user-defined library |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

The stepname within the FORVLKMM cataloged procedure is LOAD.

**Example 16:**

To create a multi-member PDS on a FILE volume and then compile and add a partially link-edited program to the PDS. The program must be fully link-edited along with all of its subroutines, as shown in the next example, before it is executed.

```
//  EXEC  PGM=IEFBR14
//NEWPDS  DD  DSN=aaaaiii.dsname,DISP=(NEW,CATLG),
//            UNIT=FILE,SPACE=(TRK,(10,2,3))
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
  (source program)
//stepname EXEC FORVLKMM,NAME='aaaaiii.dsname',
// PROGRAM=progname,OPTIONS=NCAL
```

**Example 17:**

To fully link-edit and add a VS FORTRAN program to a cataloged PDS, where one or more of the subroutines is being compiled. The same PDS is used to resolve external references; therefore, LIBNAME and NAME refer to the same data set.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
  (source program)
//stepname EXEC FORVLKMM,LIBNAME='aaaaiii.dsname',
// NAME='aaaaiii.dsname',PROGRAM=progname
//LOAD.SYSLIN  DD
//  DD  *
    INCLUDE  SYSLIB(main progname)
    ENTRY MAIN
```

The INCLUDE and ENTRY statements are control statements to the Binder. They always begin after column 1. The INCLUDE statement is used to define as input to the Binder modules that would not automatically be brought in. The ENTRY statement indicates the starting point of the program. The entry name is always MAIN for FORTRAN. These control statements and the two preceding DD statements are not needed in this example if the main program is one of the routines being compiled. In general, the ENTRY statement is not needed for FORTRAN if the main program is the first input to the Binder or if it is in object module form.

**Example 18:**

To fully link-edit and add a VS FORTRAN main program or subroutine to a PDS, where the main program and its subroutines were previously stored in the same PDS as partially link-edited load modules. If ‘progname’ and ‘main progname’ are the same, the partially link-edited main program will be replaced.

```
//stepname  EXEC  FORVLKMM,NAME=‘aaaaiii.dsname’,
//  PROGRAM=progname,
//  LIBNAME=‘aaaaiii.dsname’
//LOAD.SYSLIN  DD  *
    INCLUDE SYSLIB(main progname)
```

**Example 19:**

To link-edit and execute a program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname  EXEC  FORVCOMP
//COMP.SYSIN  DD  *
   (source program)
/*
//stepname  EXEC  FORVLKGO,
//  LIBNAME=‘aaaaiii.dsname1’,
//  ALTNAME=‘aaaaiii.dsname2’
//LOAD.SYSLIN  DD
//  DD  *
  INCLUDE SYSLIB(main program name)
  ENTRY entryname
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

## 6.3 Using Programs from User Libraries

The CALL procedure is used to execute a fully link-edited program. This procedure provides the user with the DD statements needed to use the printer (FT06F001 or FT15F001). The ddname SYSIN may be used in place of FT05F001. These DD statements are the same ones supplied in the LKGO procedure. All FORTRAN execution-time error messages are written to the DD statement FT06F001; because of a local system standard, this cannot be overridden. The user must supply any additional DD statements required for the proper execution of the program.

**Symbolic Parameters for FORVCALL**

| **Required** | **Value to be supplied** |
|---|---|

| | |
|---|---|
| NAME='aaaaiii.dsname' | Dsname of PDS containing load module |
| **Optional** | **Value to be supplied** |
| DISK=fileser | Volume for PDS; required only if the data set is not cataloged |
| STORAGE=type | Unit name for PDS; FILE is the default |
| PROGRAM=progname | Member name for load module; the default is MAIN |
| CORE=nnnnK | Region for GO step; 4096K is the default |

The stepname within the FORVCALL cataloged procedure is GO.

**Example 20:**

To execute a VS FORTRAN program that has been previously stored by an LKSM procedure.

```
//stepname EXEC FORVCALL,NAME='aaaaiii.dsname'
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 21:**

To execute a fully link-edited VS FORTRAN program stored in a cataloged PDS.

```
//stepname EXEC FORVCALL,NAME='aaaaiii.dsname',
//  PROGRAM=progname
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

## 6.4 Link-editing from a User Library

User-defined libraries can be specified to be searched in resolving external references. Both the Binder and the Loader offer this facility. The symbolic parameter LIBNAME defines the first such library. ALTNAME is available if it is necessary to define a second private library. These private libraries are searched in their order of concatenation; if members with duplicate names exist, the first one found will be selected. The private libraries are searched after the FORTRAN language library and before NIH.UTILITY.

## Symbolic Parameters for FORVLKGO and FORVLDGO

| **Required** | **Value to be supplied** |
|---|---|
| None | None |

| **Optional** | **Value to be supplied** |
|---|---|
| OPTIONS=parms | Binder or Loader parameters |
| CORE=nnnnK | Region for GO step; the defaults are 4096K for LKGO; 1000K for LDGO |
| LIBNAME=‘aaaaiii.dsname’ | Dsname of first user-defined library |
| LIBDISK=fileser | Volume for first library; required only if the data set is not cataloged |
| LIBSTOR=type | Unit name for first library; FILE is the default |
| ALTNAME=‘aaaaiii.dsname’ | Dsname of second user-defined library |
| ALTDISK=fileser | Volume for second library; required only if the data set is not cataloged |
| ALTSTOR=type | Unit name for second library; FILE is the default |

**Example 22:**

To use the Binder when a main VS FORTRAN program is compiled and its subroutines are stored as load modules in a private user library.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLKGO,LIBNAME=‘aaaaiii.dsname’
//GO.ddname DD etc.
```

**Example 23:**

To use the Loader when a main VS FORTRAN program program is compiled and some subroutines are stored as load modules in private user libraries.

```
//stepname EXEC FORVCOMP
//COMP.SYSIN DD *
   (source program)
//stepname EXEC FORVLDGO,
//    LIBNAME=‘aaaaiii.dsname1’,
//    ALTNAME=‘aaaaiii.dsname2’
//GO.ddname DD etc.
```

The LKGO procedure may also be used to relink-edit and execute a partially resolved load module stored in a partitioned data set.

The examples above assume the subroutines were stored using the same names they are called by. If these names are not the same, INCLUDE statements must be supplied for the subroutines.

**Example 24:**

To link-edit and execute a VS FORTRAN main program and its subroutines which have been partially link-edited and stored into a PDS.

```
//stepname  EXEC  FORVLKGO,
//  LIBNAME='aaaaiii.dsname'
//LOAD.SYSLIN  DD  *
  INCLUDE  SYSLIB(main program name)
/*
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

**Example 25:**

To link-edit and execute a VS FORTRAN program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname  EXEC  FORVCOMP
//COMP.SYSIN  DD  *
   (source program)
/*
//stepname  EXEC  FORVLKGO,
//  LIBNAME='aaaaiii.dsname1',
//  ALTNAME='aaaaiii.dsname2'
//LOAD.SYSLIN  DD
//  DD  *
  INCLUDE SYSLIB(main program name)
  ENTRY entryname
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
   (data)
```

# 7 PROGRAMMING AND RUNNING TIPS

The hints in this section apply to VS FORTRAN programs run at the NIH Computer Center.

## 7.1 Coding in FORTRAN

- The names used in the SUBROUTINE and the CALL statements referencing a subroutine must always be the same. If this name is not the same as the name under which the subroutine is stored in the online library, then a Binder INCLUDE control statement must also be used for the subroutine in the link-edit step.
- Use of the LABEL=(,,,IN) DD parameter to prevent writing over an input tape data set is not recommended because it does not work in all cases. Unless the first I/O operation is a READ or WRITE (it might be a REWIND), the tape may be written on regardless of the specification in the LABEL parameter.
- The DEBUG, SDUMP, and PDUMP facilities can be used when debugging a VS FORTRAN program to check the validity of subscripts, trace the flow of the program, print the variables whenever their values change, and print given variables at any point in the program.
- When using FORTRAN to create variable length spanned records (all unformatted I/O in FORTRAN uses VS or VBS records), the LRECL specified must be as long as the longest record. Otherwise, a warning message, AFB201, will appear stating that the file may not be usable by other programs. Once migrated, such data sets cannot be retrieved.
- At the NIH Computer Center, a program will terminate with a completion code of 16 after receiving only one AFB219 execution time error if the file “opened” or “closed” does not exist. This differs from the standard described in IBM documentation that indicates that 10 such errors may occur before program termination.
- Unlike the indications in IBM documentation, not all AFB219 errors are caused by DD statement errors. Requesting insufficient REGION for the program can also cause this error. As indicated in the IBM documentation, this error will occur 10 times before program termination.
- Some programming errors normally expected to cause an ABEND in a FORTRAN program are treated differently by FORTRAN. For example, ABENDs which might occur when a data set is opened (such as an 813 if a tape data set name is misspelled) are intercepted by FORTRAN execution-time library routines and a completion code of 16 is issued instead.

  Users who code conditional EXEC statements (using the COND parameter) should be mindful of this ABEND-handling by executing FORTRAN programs and adjust their condition code tests accordingly. For example, COND=EVEN and COND=ONLY, which apply only to system ABEND completion codes, would not necessarily detect a problem in the execution of a FORTRAN load module. It may be advisable to test the jobstep completion code of the executing FORTRAN load module for 16, or greater than zero.

## 7.2 Using Data Set Reference Numbers

- For the VS FORTRAN compiler, the data set reference numbers may range from 0-99 (i.e., FT00F001-FT99F001). The data set reference number nn (FTnnF001) corresponds to the READ (**nn**, formatno) or WRITE in the user's FORTRAN programs.
- VS FORTRAN load module execution time messages are printed on data set reference number 6 (FT06F001). FT06F001 is opened for output by FORTRAN because it is the default ddname for all AFB type diagnostics and execution time messages. The DD statement for unit 6 should always be coded as SYSOUT=A, e.g.,

  ```
  //FT06F001 DD SYSOUT=A
  ```

  Never use unit 6 for input; this is an error that will cause unpredictable results.
- The VS FORTRAN additional facilities, which permit I/O operations without data set reference numbers, are defined by the Computer Center such that:

| | |
|---|---|
| READ format,list | reads data set reference number 5 (FT05F001 or SYSIN) |
| PRINT format,list | writes data set reference number 6 (FT06F001) |

## 7.3 Writing Efficient Programs

- Consult the section entitled "Optimizing Your Program" in *the VS FORTRAN Version 2: Programming Guide for CMS & MVS* for a wealth of information on efficient VS FORTRAN coding techniques.
- Avoid using subroutines and functions for small repeated tasks. Use statement functions for short functions instead of calling external subroutines. Statement functions are compiled directly in the module.
- Apply the information gained during analysis of the problem before programming to simplify the problem and speed up the numerical procedure.
- Arrange the program logic to avoid branches whenever possible.
- Make the most probable result of all logical IF statements a simple drop-through instead of a branch.
- Reduce I/O to the minimum necessary when debugging is complete.
- Because of round-off error in low-order bits, do not test for equal using floating-point variables; use .GE. or .LE.
- Use SQRT instead of **.5. For small powers, use A*A*A ... or A**I with I = R instead of A**R where R is a floating-point integer. (Values raised to integer powers are computed by repetitive multiplication, whereas values raised to real powers are computed using logarithmic and exponential routines. A**R can be forty times slower than A**I).

- Calculate all quantities that are constant throughout a program at the beginning, and calculate all quantities constant throughout a loop outside the loop. For example:

```
          DO 20 I = 1,450
     20   C(I+3,2*I+1) = (D*(I+2))**(2*K)+E-2.*L-1
```

  should be written

```
          M = 2*K
          F = E-2.*L-1
          DO  20  J=3,452
     20   C(J+1,2*J-3) = (D*J)**M+F
```

- If several tests or operations are to be done in DO loops, incorporate as many as possible in a single DO loop rather than setting up separate DO loops for each test or operation.
- Code the first subscript in the innermost loop of the program. For example:

```
          DO 500 J=1,500
          DO 500 I=1,500
     500  TOTAL=TOTAL+ARRAY(I,J)
```

- When using CHARACTER data, relatively long character variable lengths (CHARACTER*20 or more) should be used. This reduces or eliminates the need for long, costly DO LOOPS to initialize data.

## 7.4 Increasing I/O Efficiency

- If your program reads or writes portions of arrays, the use of EQUIVALENCE statements will permit the use of simple array names in I/O statements. For example:

```
       INTEGER BUFFER(1000),BUFF1(500),BUFF2(500)
       EQUIVALENCE (BUFFER(1),BUFF1(1)),
      *(BUFFER(501),BUFF2(1))
     1 WRITE(3)INUM,(BUFFER(I),I=1,500)
     2 WRITE(3)INUM,BUFF1
       RETURN
       END
```

  WRITE statements 1 and 2 are equivalent in effect, but the second requires less CPU time.

- The I/O of a list of many variables can be made more efficient in the following manner. Use a COMMON or named COMMON statement to place the variables in contiguous storage spaces. Define a singly dimensioned array the length of the list of variables and use an EQUIVALENCE statement to cause the array to reference the same storage locations as the list of variables. Then use the array name for I/O. An example is:

```
COMMON/LIST/A(20),B(30),C,D,E(15),F,G(25)
DIMENSION XOUT(93)
EQUIVALENCE (A(1),XOUT(1))
READ (5,100) XOUT
.
.
WRITE (6,200) XOUT
```

## 7.5 Data Formats for Inter-Language Communication

The following figures show the ways data can be stored. The source language definitions for each data type are given under the COBOL, FORTRAN, and PL/I headings. For more specific information on data formats, consult the appropriate language manuals and *the IBM ESA/390 Principles of Operation*, SA22-7201.

The "MACHINE DATA FORMAT" column in the figures below shows a bit breakdown of the data type as stored internally. Bit positions are written vertically under the machine data format symbols they refer to.

**CHARACTER**

| COBOL | FORTRAN | PL/I | TYPE |
|---|---|---|---|
| PIC X(n) DISPLAY<br><br>1<=n<=32767 | CHARACTER*n<br><br>1<=n<=3267 | CHAR(n)<br><br>1<=n<=32767 | Length = n bytes |

| MACHINE DATA FORMAT | | | | EXAMPLE | |
|---|---|---|---|---|---|
| Char 1 | Char 2 | … | Char n | Value | Internal hex representation |
| 0 0<br>-<br>0 7 | 0 1<br>-<br>8 5 | | | ABCD | C1C2C3C4 |

**Figure 3. Character Formats for Inter-Language Communication**

**FIXED POINT**

The fixed point two-word data type, which is available only in COBOL, is simulated through software and requires all data items to be aligned on a word boundary.

The "Range" given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

Assumed decimal points in COBOL and PL/I are not shown in the table. They are stored in the same way as other numbers; instructions generated by the compilers keep track of the position of the assumed decimal point.

| COBOL | FORTRAN | PL/I | TYPE |
|---|---|---|---|
| PIC S9(1-4) COMP (or COMP-4)<br><br>Range: -9999 to 9999 | INTEGER*2<br><br>Range: -32768 to 32767 | FIXED BIN (1-15,0)<br><br>Range: -32768 to 32767 | Halfword<br><br>Length = 2 bytes. |
| PIC S9(5-9) COMP (or COMP-4)<br><br>Range: -(9)9s to +(9)9s | INTEGER*4<br><br>Range: -2147483648 to 2147483647 | FIXED BIN 16-31,0)<br><br>Range: -2147483648 to 2147483647 | Fullword<br><br>Length =4 bytes. |
| PIC S9(10-18) COMP (or COMP-4)<br><br>Range: -(18)9s to +(18)9s | ----- | ----- | Two-word<br><br>Length = 8 bytes. |

**Figure 4. Fixed Point Formats for Inter-Language Communication**

| MACHINE DATA FORMAT | EXAMPLES | |
|---|---|---|
| | Value<br>---------- | Internal hex representation<br>-------------------- |
| S I<br>0 0 - 1<br>0 1 5 Halfword | +1234<br>-1234 | 04D2<br>FB2E |
| S I<br>0 0 - 3<br>0 1 1 Fullword | +1234<br>----------<br>-1234 | 000004D2<br>---------------------<br>FFFFFB2E |
| S I<br>0 0 - 6<br>0 1 3<br>Two-word | +1234<br>----------<br>-1234 | 0…04D2<br>---------------------<br>F...FB2E |

“S” is a binary sign bit: 0 is positive; 1 is negative.
“I” is a 15, 31, or 63 bit integer.
**Figure 4** (Continued)

**FLOATING POINT**
Magnitude is the range of a number expressed in powers of ten.

Although the numbers are represented the same internally, peculiarities in languages reduce the precision of numbers in some cases. The degree of precision given in the table is good in all cases. Fractional precisions occur because of the difference between the decimal representation and the machine’s internal storage of numbers.

| COBOL | FORTRAN | PL/I | TYPE |
|---|---|---|---|
| COMP-1<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>7.2 digits | REAL*4<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>7.2 digits | FLOAT<br>DEC(1-6)<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>6 digits | Short<br><br>Length = 4 bytes |
| COMP-2<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>16 digits | REAL*8<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>16.8 digits | FLOAT<br>DEC(7-16)<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>16 digits | Long<br><br>Length = 8 bytes |
| ------ | REAL*16<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>35 digits | FLOAT<br>DEC(17-33)<br><br>Magnitude:<br>10**-78 to 10**75<br>Precision:<br>33 digits | Extended<br><br>Length = 16 bytes |

**Figure 5. Floating Point Formats for Inter-Language Communication**

| MACHINE DATA FORMAT | EXAMPLES | |
|---|---|---|
| | Value | Internal hex representation |
| S \| E \| F<br>0 0-0 0 - 3<br>0 1 7 8 1<br>Short | -----------<br>+1234<br>-----------<br>-1234 | ---------------------<br>434D2000<br>---------------------<br>C34D2000 |
| S \| E \| F<br>0 0-0 0 - 6<br>0 1 7 8 3<br>Long | +1234<br>-----------<br>-1234 | 434D20…0<br>---------------------<br>C34D20…0 |
| S \| E \| F<br>0 0-0 0 - 6<br>0 1 7 8 3<br>\| F (continued)<br>0 - 0 0 6<br>0 7 8 3<br>Extended | +1234<br>-----------<br>-1234 | 434D20…0<br>---------------------<br>C34D20…0 |

"S" is a binary sign bit: 0 is positive; 1 is negative.
"E" is a seven bit exponent with a value between hex 16** -64 and 16** +63.
"F" is a fraction, which may be 24, 56, or 112 bits long.
**Figure 5** (Continued)

## ZONED DECIMAL

The “Range” given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

| COBOL | FORTRAN | PL/I | TYPE |
|---|---|---|---|
| PIC 9(n)<br>DISPLAY<br><br>1<=n<=18<br><br>Range: 0 to (18)9s | ------ | PIC ‘(n)9’<br><br>1<=n<=15<br><br>Range: 0 to (15)9s | Unsigned<br><br>Length = n bytes. |
| PIC S9(n)<br>DISPLAY<br><br>1<=n<=18<br><br>Range:<br>-(18)9s to +(18)9s | ------ | PIC ‘(n-1)9T’<br><br>1<=n<=15<br><br>Range:<br>-(15)9s to +(15)9s | Signed<br><br>Length = n bytes. |

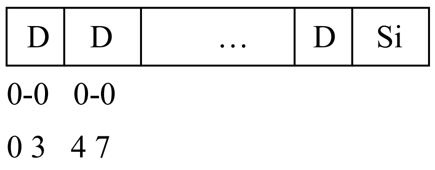
| MACHINE DATA FORMAT | EXAMPLES | |
|---|---|---|
| | Value | Internal hex representation |
| | ---------- | --------------------- |
| Z \| D \| Z \| D \| … \| Z \| D<br>0-0 0-0 0-1 1-1<br>0 3 4 7 8 1 2 5<br>Unsigned | 1234 | F1F2F3F4 |
| Z \| D \| Z \| D \| … \| Si \| D<br>0-0 0-0 0-1 1-1<br>0 3 4 7 8 1 2 5<br>Signed | +1234<br>----------<br>-1234 | F1F2F3C4<br>---------------------<br>F1F2F3D4 |

“Z” is a 4 bit zone code with a value of hex F.
“D” is a 4 bit binary decimal number with a value between hex 0 and 9.
“Si” is a 4 bit sign code: A, C, E, and F are positive; B and D are negative.

**Figure 6. Zoned Decimal Formats for Inter-Language Communication**

**PACKED DECIMAL**
The "Range" given in the table indicates the minimum and maximum values numbers can have in all uses of the language. Idiosyncrasies in languages reduce the full range of numbers in some cases even though they are represented the same internally.

| COBOL | FORTRAN | PL/I | TYPE |
|---|---|---|---|
| COMP-3<br>PIC 9(n)<br><br>1<=n<=18<br><br>Range: -(18)9s to<br>+(18)9s | ------ | FIXED<br>DEC(n)<br><br>1<=n<=15<br><br>Range: -(15)9s to<br>+(15)9s | Length in<br>bytes = (n+1)/2<br>rounded up. |

| MACHINE DATA FORMAT | EXAMPLES | |
|---|---|---|
|  | Value | Internal hex representation |
| | --------------- | --------------------- |
| | -1234 | 01234C |
| | --------------- | --------------------- |
| | -1234 | 01234D |

"D" is a 4 bit binary decimal number with a value hex 0 through 9.
"Si" is a 4 bit sign code: A, C, E, and F are positive; B and D are negative.

**Figure 7. Packed Decimal Formats for Inter-Language Communication**

## 7.6 Dynamic Data Set Allocation

Normally, when FORTRAN programs are run at the NIH Computer Center the JCL specifies the various input and output data sets with DD statements. FORTRAN allows users to specify these data sets dynamically within the program.

For example:

```
//  EXEC  FORVCOMP C
C This job dynamically creates a data set called
C AAAAIII.FORT.TEST on a public FILE volume.  If the data
C set does not exist, this is the equivalent to
C specifying the following DD statement:
C
C   //GO.FT10F001  DD  DSN=AAAAIII.FORT.TEST,
C   //   DISP=(NEW,CATLG),
C   //   UNIT=FILE,SPACE=(TRK,(5,5)),
C   //   DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
C
C If the data set exists before the job is submitted,
C this program is equivalent to coding:
C
C   //GO.FT10F001  DD  DSN=AAAAIII.FORT.TEST,
C   //   DISP=(OLD,KEEP),
C   //   SPACE=(TRK,(5,5))
C
         CALL FILEINF(IRC, ‘RECFM’, ‘FB’, ‘LRECL’, 80,
     X   ‘BLKSIZE’,8000, ‘TRK’, 5, ‘SECOND’, 5,
     X   ‘DEVICE’,’FILE’)
         OPEN(10,FILE=‘/AAAAIII.FORT.TEST’)
         WRITE(10,100)
100      FORMAT(‘TEST DATA’)
         STOP
         END
//  EXEC  FORVLKGO
```

See *VS FORTRAN Version 2: Language and Library Reference*, SC26-4221 for details on the use of the FILEINF function that is used to set up the characteristics of the data set being allocated.

# INDEX

# *Using FORTRAN at the NIH Computer Center*

**Document Evaluation**

**Is the Manual:**

| | YES | NO |
|---|---|---|
| Clear? | ☐ | ☐ |
| Well organized? | ☐ | ☐ |
| Complete? | ☐ | ☐ |
| Accurate? | ☐ | ☐ |
| Suitable for the beginner? | ☐ | ☐ |
| Suitable for the advanced user? | ☐ | ☐ |

**Comments:**

Please give page references where appropriate. If you wish a reply, include your name and mailing address.

Send to: Application Services Branch
Division of Computer System Services, CIT
National Institutes of Health
Building 12A, Room 4011
Bethesda, MD 20892-5607

FAX to: (301) 496-6905

ICD or Agency:
Date Submitted:
Name (Optional):
E-Mail Address:

Revision date: 9/98