Using Assembler Language at the NIH Computer Center

September 1998

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

Publication No. CIT182

Table Of Contents

1 INTRODUCTION 1

1.1 Procedure Names 2

2 DCB INFORMATION FOR SYSOUT DATA SETS 4

3 HIGH LEVEL ASSEMBLER LANGUAGE PUBLICATIONS 5

4 HIGH LEVEL ASSEMBLER COMPILER OPTIONS 8

5 ASSEMBLING AND RUNNING PROGRAMS 9

5.1 Using the Assembler 9
5.2 Using the Binder 10
5.3 Creating and Using Object Modules 12
5.4 Using the Loader 15

6 STORING AND USING PROGRAMS IN USERS LIBRARIES 17

6.1 Storing Programs in Single-Member User Libraries 17
6.2 Storing Programs in Multi-Member User Libraries 20
6.3 Using Programs from User Libraries 23
6.4 Using the Binder from a User Library 23

7 PROGRAMMING AND RUNNING TIPS 27

7.1 Modular Programming 27
7.2 Program Commenting 27
7.3 Standard Binder Conventions 28
7.4 Register Usage 28
7.5 Some Coding Hints 29
7.5.1 General Programming Guidelines 29
7.5.2 Instruction Usage Hints 30
7.6 Data Formats for Inter-Language Communication 34

1 INTRODUCTION

This manual describes the use of the Assembler programming language at the NIH Computer Center. This manual is intended to give programmers the Assembler 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 *Computer Center User's Guide*, *Batch Processing and Utilities at NIH*, and the manuals described in Section 3 of this publication.

The Assembler programming language receives full (Level 1) support. Questions on Assembler 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 through WYLBUR's ENTER PTR command.

Changes that affect the use of Assembler 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*.

The IBM Operating System Assembler is a symbolic programming language used to write programs for the MVS System. The language provides a convenient means for representing the machine instructions and related data necessary to program the MVS System. The NIH Computer Center uses the IBM High Level Assembler program product.

No Federal Information Processing Standard (FIPS) has been established for this language. Programs written in languages meeting FIPS standards can be transported more readily between federal installations and different vendors' mainframes. Federal policy encourages the use of languages with FIPS standards.

1.1 Procedure Names

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

The procedure names for Assembler Language are:

ASMHCOMP
ASMHOBJ
ASMHLKGO
ASMHLDGO
ASMHLKMM
ASMHLKSM
ASMHCALL

Each procedure name follows the pattern:

lllvffff

where	"lll"	is the language prefix (ASM for Assembler)
	"v"	is the version (is the version (H))
	"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 SYSOUT DATA SETS

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

PROCEDURE NAME	STEP NAME	DD NAME	DEFAULT RECFM/BLKSIZE
ASMHCOMP	COMP	SYSTERM SYSPRINT SYSUDUMP	FM 121 FM 121 VBA 125
ASMHLDGO	GO	SYSLOUT	FA 121
ASMHLKGO	LOAD	SYSPRINT SYSUDUMP	FA 121 VBA 125
ASMHLKMM	LOAD	SYSPRINT SYSUDUMP	FA 121 VBA 125
ASMHLKSM	LOAD	SYSPRINT SYSUDUMP	FA 121 VBA 125
ASMHOBJ	COMP	SYSTERM SYSPRINT SYSUDUMP	FM 121 FM 121 VBA 125

Figure 1. SYSOUT DCB Information for Assembler Procedures

3 HIGH LEVEL ASSEMBLER LANGUAGE PUBLICATIONS

The CIT Technical Information Office distributes general information, technical and vendor publications and certain software to the user community. *Using Assembler Language 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 Assembler can be ordered:

IBM High Level Assembler Language Reference, GC26-4940

This publication is a reference manual for the programmer using High Level Assembler. It will enable the user to answer specific questions about language functions and specifications.

IBM High Level Assembler Programmer's Guide, SC26-4941

This publication tells how to use High Level Assembler. It provides the guide to assembling, link editing, running, interpreting listings and programming considerations. Only Parts 1 and 2 are applicable to the MVS operating systems. Part 3 is for VM/CMS only. Part 4 is for VSE.

IBM High Level Assembler General Information, GC26-4943

This manual introduces the major features of High Level Assembler to help users decide whether this programming language meets their data processing needs.

IBM MVS/ESA Principles of Operation, SA22-7200

This manual is the machine reference manual for the IBM MVS extended system architecture mode. It is written principally for Assembler Language programmers to be used in conjunction with the appropriate Assembler Language manuals.

IBM MVS/ESA Programming: Assembler Services Guide, GC28-1466

This publication shows applications programmers how to use the services of the supervisor, the macro instructions used to request these services, and the linkage conventions used by the control program to provide these services.

IBM MVS/ESA Programming: Assembler Services Reference, GC28-1474

This publication describes some of the macros (or macro instructions) that the system provides. These macros are available to any Assembler Language program. Programmers can use these macros to invoke the system services that they need. This publication includes the detailed information; such as the function, syntax, and parameters, needed to code the macros.

DFSMS/MVS Program Management, SC26-4916

This publication is intended to help you learn about and use the program management functions provided by DFSMS/MVS. DFSMS/MVS works in conjunction with MVS/ESA SP to manage programs, performing the steps necessary to create and execute programs on the system. These functions are performed by various DFSMS/MVS program management components, including the program management binder, the program management loader, the Binder, the batch loader, and the transport utility.

IBM DFSMS/MVS Macro Instructions for Data Sets, SC26-4913

This publication contains descriptions and definitions for the data management macro instructions available in Assembler Language. No information is included for VSAM.

IBM MVS/ESA JCL User's Guide, GC28-1473

This publication is designed as a user's guide to be used when deciding how to perform job control tasks. It does not describe how to code the statements.

IBM MVS/ESA JCL Reference, GC28-1479

This publication is designed as a reference book to be used while coding JCL statements.

MVS/ESA System Messages, *Volume 1* (GC28-1480), *Volume 2* (GC28-1481), *Volume 3* (GC28-1482), *Volume 4* (GC28-1483), and *Volume 5* (GC28-1484)

These publications list the systems messages produced by IBM-supplied components of the operating system. The causes of the messages are explained, the accompanying actions by the operating system are described, and appropriate responses are suggested.

IBM MVS/ESA JES2 Messages, GC23-00085

This publication documents the messages issued by the MVS/ESA operating system.

IBM MVS/ESA Systems Codes, GC28-1486

This publication lists the system completion codes and wait state codes issued by the MVS/ESA operating system. Each code is explained, and where appropriate, a programmer response is indicated.

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 HIGH LEVEL ASSEMBLER COMPILER OPTIONS

The NIH Computer Center has customized the following options for High Level Assembler:

TERM	The Assembler will write diagnostic information on the device specified by the SYSTERM DD statement (SYSOUT=A).
XREF(SHORT)	Cross reference table of all symbols referenced in the assembly is produced.

For a complete list of all options for High Level Assembler, see the *IBM High Level Assembler Programmer's Guide*, SC26-4941.

If any options will 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 ASSEMBLING AND RUNNING PROGRAMS

The procedures in this section are used to assemble, fully resolve (using the Binder), and execute Assembler Language programs.

5.1 Using the Assembler

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

Symbolic Parameters for ASMHCOMP

Required	**Value to be supplied**
None	None

Optional	**Value to be supplied**
OPTIONS=parms	Assembler parameters
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined macro 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 macro 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 ASMHCOMP procedure is COMP.

Example 1:

To assemble source code only.

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

Example 2:

To assemble source code containing user-defined macros that are located in the user's macro library which resides on the MSS.

```
//stepname  EXEC ASMHCOMP,LIBNAME='aaaaiii.dsname'
//COMP.SYSIN  DD  *
(source program)
```

Example 3:

To assemble source code, printing the full cross-reference table and keeping the source symbol table (SYM) in the object module for later use by TSO TEST. The TEST option must be specified for both the Assembler and the Binder. This option must not be specified if the program is to be executed as a batch job (using ASMHLKGO, ASMHLDGO, or ASMHCALL).

```
//stepname  EXEC  ASMHCOMP,OPTIONS='XREF(FULL),TEST'
//COMP.SYSIN  DD  *
(source program)
```

Use of the TEST facility is described in the *TSO Extensions Command Language Reference*, SC28-1881.

5.2 Using the Binder

The LKGO procedure performs the following:

- fully resolves the program to prepare a load module for execution
- executes the load module

It provides the user with the DD statements needed to use the printer (SYSOUT) and the SORT/MERGE messages data set (SORTMSGS). The user must provide additional JCL for any I/O units (data sets) used. Section 6.4 discusses specifying user-defined libraries with LKGO.

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. If more than one DD statement is being overridden, the override statements must

be in the same order as the existing DD statements in the procedure. See the manual *Batch Processing and Utilities at NIH* for a description of the format of DD statements.

Symbolic Parameters for ASMHLKGO

Required	**Value to be supplied**
None	None

Optional	**Value to be supplied**
OPTIONS=parms	Binder parameters
CORE=nnnK	Region for GO step; 512K 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 stepnames within the ASMHLKGO cataloged procedure are LOAD for the Binder step and GO for the run step.

Example 4:

To assemble the main program and execute it. The OPTIONS parameter in the run step requests the Binder option XREF.

```
//stepname  EXEC  ASMHCOMP
//COMP.SYSIN  DD  *
(source program)
//stepname  EXEC  ASMHLKGO,OPTIONS=XREF
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
(data)
```

Example 5:

To assemble the main program and one subroutine and execute it. The CORE parameter supplies a region size for the GO step.

```
//stepname EXEC ASMHCOMP
//COMP.SYSIN DD *
(source for main program)
//stepname  EXEC  ASMHCOMP
//COMP.SYSIN  DD  *
(source for subroutine)
//stepname  EXEC  ASMHLKGO,CORE=nnnK
//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 assemble 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 fully resolve and execute the object module(s) created by the OBJ procedure(s).

Symbolic Parameters for ASMHOBJ

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	Assembler parameters
STATUS=status	Specifies whether the output data set is old or new; NEW is the default
SIZE=primary	Primary space allocation for object module; default is 500 (enough for approximately 500 source statements)
UNITS=type	Allocation units for object module; the default is blocks of 1024 bytes
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined macro 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 macro 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 ASMHOBJ procedure is COMP.

Example 6

To assemble and save the object module.

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

Example 7:

To assemble and save into an existing data set. Former contents will be destroyed.

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

Example 8:

To assemble and save overriding the default for primary space allocation. If there are more than 500 source statements, the 'primary' value should be roughly equal to the number of statements in the program.

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

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

Example 9:

To assemble, fully resolve, and run. The object module saved as "aaaaiii.dsname1" from the OBJ procedure is used as input for the Binder step.

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

Example 10:

To execute a 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 insure that the main program is defined first.

```
//stepname EXEC ASMHLKGO
//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 11:

To execute a program where the main program is to be assembled 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 ASMHCOMP
//COMP.SYSIN DD *
(source program)
//stepname EXEC ASMHLKGO
//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 Binder 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. DD statements are provided for use of the printer (SYSOUT).

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 can be found in the manual *Batch Processing and Utilities at NIH*.

Symbolic Parameters for ASMHLDGO

Required	**Value to be supplied**
None	None
Optional	**Value to be supplied**
OPTIONS=parms	Loader and GO parameters
CORE=nnnK	Region for GO step; 300K is the default
EPT=entry	Entry point for the main program; no default is supplied
LIBNAME='aaaaiii.dsname'	Dsname of first user-defined macro 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 macro 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 ASMHLDGO procedure is GO.

Example 12:

To assemble the main program and execute it. The OPTIONS parameter in the run step requests the Loader option XREF.

```
//stepname  EXEC  ASMHCOMP
//COMP.SYSIN  DD  *
(source program)
//stepname  EXEC  ASMHLDGO,OPTIONS=XREF
//GO.ddname  DD  etc. (as many as needed)
//GO.SYSIN  DD  *  (if needed)
(data)
```

6 STORING AND USING PROGRAMS IN USERS LIBRARIES

The following procedures were developed to store user programs in load module form. Each user can develop and maintain 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 a B37 or D37 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 resolved. 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 resolved and is, therefore, directly executable without being resolved again. If the external references and calls are not to be resolved, the NCAL option must be specified in the EXEC statement for the procedure used to store the program. The program is then partially resolved and must be reprocessed by the Binder before it can be executed.

Executing fully resolved load modules may cost less because a Binder 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 Binder step each time the program is run.

6.1 Storing Programs in Single-Member User Libraries

The LKSM procedure is used to fully resolve and store a load module (output of the Binder) 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 Binder step, deletes the PDS if it already exists on the specified disk. Then the Binder 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. If the output library is to be created on the MSS, STORAGE=MSS must be specified. Additionally, the symbolic parameters SIZE, UNITS, and INCR must be coded with the appropriate values for requesting space on the MSS.

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 before NIH.UTILITY.

Symbolic Parameters for ASMHLKSM

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 ASMHLKSM procedure are SCRATCH, for the step to scratch the data set if it already exists, and LOAD for the Binder step.

Example 13:

To assemble, fully resolve, and store a program into a single-member PDS.

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

Example 14:

To assemble, fully resolve, and store a 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 parameter allows the user to obtain at least 10 tracks for the load module (unneeded space is released).

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

Example 15:

To fully resolve and store a main program and subroutines 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 ASMHLKSM,NAME='aaaaiii.dsname',
//  DISK=fileser
//LOAD.SYSLIN DD DSN=aaaaiii.dsname1,DISP=SHR
// DD DSN=aaaaiii.dsname2,DISP=SHR
// DD DSN=aaaaiii.dsname3,DISP=SHR
```

Example 16:

To assemble a main program (ASMHCOMP), fully resolve it using subroutines previously assembled with the OBJ procedure, and create a fully resolved single-member load module (ASMHLKSM).

```
//stepname EXEC ASMHCOMP
//COMP.SYSIN DD *
(source program)
//stepname EXEC ASMHLKSM,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 *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 before NIH.UTILITY. If no libraries are to be searched (no external references are to be resolved), OPTIONS=NCAL must be specified for the ASMHLKMM step; this creates a partially resolved load module.

Symbolic Parameters for ASMHLKMM

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
LTSTOR=type	Unit name for second library; FILE is the default

The procedure name is ASMHLKMM. The stepname within the cataloged procedure is LOAD.

Example 17

To create a multi-member PDS on a FILE volume and then assemble and add a partially resolved program to the PDS. The program must be fully resolved 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 ASMHCOMP
//COMP.SYSIN DD *
(source program)
//stepname EXEC ASMHLKMM,NAME='aaaaiii.dsname',
// PROGRAM=progname,OPTIONS=NCAL
```

Example 18

To fully resolve and add a program to a multi- member PDS, where the main program and its subroutines were previously stored in the same PDS as partially resolved load modules. If 'progname' and 'main progname' are the same, the partially resolved main program will be replaced.

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

Example 19:

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

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

The INCLUDE and ENTRY control statements are passed 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. If the ENTRY statement is not provided and no entry point is specified in the Assembler program, the first byte of the first control section of the load module will be used as the entry point.

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 assembled. In general, the ENTRY statement is not needed for Assembler Language if the main program specified an entry point and is the first input to the Binder or is in object module form.

Example 20:

To fully resolve and execute a program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname  EXEC  ASMHCOMP
//COMP.SYSIN  DD  *
(source program)
/*
//stepname  EXEC  ASMHLKGO,
//  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 resolved program. This procedure provides the user with the DD statements needed to use the printer (SYSOUT) and the SORT/MERGE messages data set (SORTMSGS). These DD statements are the same ones supplied in the ASMHLKGO procedure. The user must supply any additional DD statements required for the proper execution of the program.

Symbolic Parameters for ASMHCALL

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=nnnK	Region for GO step; 500K is the default

The internal stepname for the ASMHCALL procedure is GO.

Example 21:

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

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

Example 22:

To execute a fully resolved program previously stored in a PDS on the MSS.

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

6.4 Using the Binder 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 before NIH.UTILITY. if members with duplicate names exist, the first one found will be selected. The private libraries are searched before NIH.UTILITY.

Symbolic Parameters for ASMHLKGO and ASMHLDGO

Required	**Value to be supplied**
None	None

Optional	**Value to be supplied**
OPTIONS=parms	Binder or Loader parameters
CORE=nnnK	Region for GO step; 512K is the default
EPT=entry	Entry point for main program (Loader only); no default is supplied
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 23:

To use the Binder when the main program has been assembled and its subroutines are stored as load modules in a private user library on one of the FILE disks.

```
//stepname EXEC ASMHCOMP
//COMP.SYSIN DD *
(source program)
//stepname EXEC ASMHLKGO,LIBNAME='aaaaiii.dsname'
//GO.ddname DD etc.
```

Example 24:

To use the Loader when the main program has been assembled and its subroutines are stored as load modules in a private user library on the MSS.

```
//stepname EXEC ASMHCOMP
//COMP.SYSIN DD *
(source program)
//stepname EXEC ASMHLDGO,LIBNAME='aaaaiii.dsname'
//GO.ddname DD etc.
```

The LKGO procedure may also be used to fully re-resolve 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 25:

To fully resolve and execute a main program and its subroutines that have been partially resolved and stored into a PDS.

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

Example 26:

To fully resolve and execute a program where the main program and some subroutines are in two separate PDSs and other subroutines are being compiled.

```
//stepname  EXEC  ASMHCOMP
//COMP.SYSIN  DD  *
(source program)
//stepname  EXEC  ASMHLKGO,
//  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 suggestions in this section apply to Assembler Language jobs run at NIH.

7.1 Modular Programming

Modular programming may be defined as a method of dividing problem solutions into logical parts so that they may be solved by arranging programs into sections which are easily understood and written. The result of this method of program design is a group of related routines controlled by a single routine called a “main line” or “driver” program. Using this method, routines may be added, deleted, or modified without affecting the remainder of the program; routines may be used in several different places within a program without duplicate coding; and sections may be coded and tested independently. The primary objectives of modular programming are an increase in ease of understanding and modification and standardization of program organization. When writing modular programs:

- Each routine should establish its own base register for addressability.
- All called routines should return to an address supplied by the calling routine.
- Standard OS/VS Binder conventions should be followed.

7.2 Program Commenting

One of the most useful tools available for maintaining programs is the existence of meaningful comments on a program listing. Comments should be written to explain what takes place in a program rather than to act as extraneous words on a listing. It is very helpful to precede programs with commentary explaining:

- purpose of the routine
- the action that takes place and the method used
- conditions assumed or expected upon entry; i.e., register contents, parameter list structure, etc.
- conditions set upon exiting from the routine
- other routines that call and are called by this routine

In addition to prologue commentary, instruction coding should be documented. These comments may point out restrictions, techniques, cautions, etc. that one should know before attempting to modify the program. For ease of reading, all in-line commentary should start in the same column.

7.3 Standard Binder Conventions

It is the responsibility of a calling routine, whether the user's program or the operating system, to:

- insert the entry-point address of the called routine in general register 15
- insert the return address in general register 14
- insert the address of an 18 word save area in general register 13
- insert the address of a parameter list (if any) in general register 1

It is the responsibility of the called routine to:

- save the contents of all of the caller's general registers upon entry
- restore the contents of the caller's general registers immediately before returning
- return to the address supplied by the calling routine

A detailed description of OS/VS Binder conventions, save areas, and parameter lists may be found in the publication, *IBM MVS/ESA Programming: Assembler Services Guide*, GC28-1466

7.4 Register Usage

General registers 0,1,13,14, and 15 are known as the linkage registers and are used in a prescribed manner by the Operating System. These registers should be used in the same manner by the problem program in order to avoid the possibility of register clobbering.

Registers 0,1,14, and 15 may be altered by the Operating System when using system macro instructions. Registers 2-13 remain unchanged.

Registers 0 and 1 are used to pass parameters or parameter lists between programs.

Register 13 contains the address of the save area the user provided. This save area may be used by the Operating System for any function the program requests.

Register 14 contains the return address of the program that called the user's routine, or an address within the operating system to which it is to return when it has finished processing.

Register 15 contains an entry point address when control is passed to a program from the Operating System. The entry point should be in register 15 when control is passed to another program or subroutine. Register 15 is also used to pass a return code to the calling program. When control is returned to OS, the contents of register 15 will be the condition code returned for the job step.

7.5 Some Coding Hints

The purpose of this section is to define some general rules and unusual uses of instructions that will help the programmer conserve storage and CPU time.

7.5.1 General Programming Guidelines

- At no time should a program be written which modifies the contents of an instruction during program execution. This makes maintenance unduly burdensome and may create model dependent code since the manner of instruction decoding and execution varies among different models of System 370. Instead, the EX (execute) instruction can be used to alter an instruction during execution.
- Whenever possible, arithmetic and internal data representation should be done in fixed point binary rather than packed or zoned decimal. Decimal instructions and data require more storage and CPU execution time.
- Programs should be planned to keep data movement to a minimum. Data movement can often be avoided by passing an address in a register rather than the actual data from routine to routine.
- When defining constants, place items requiring double-word alignment first, followed by full-word items, followed by half-word items, followed by items that do not require any boundary alignment.
- Do not attempt to capitalize on unique characteristics of individual operating system implementations. Programs written to take advantage of specific hardware or software features tend to be much more difficult to test and maintain.
- Emphasize reduction of short term storage requirements even though the total size of the program may increase slightly. The primary objective is to shrink and stabilize the working set of each significant phase of the program. Although size limits will always be enforced, the total size of the program is of less consequence since only the active portion of the program requires real storage.
- Optimize the main line of the program for the normal case. Remove all exception and error handling routines from the normal program flow. This will increase the density of reference of the most heavily used pages. However, these conditions should be detected in the main line to avoid unnecessary entry into another page. The actual error routines should be grouped together in pages of their own since under normal circumstances they will never be entered. Low use code that is not exception handling code should be inline (i.e., housekeeping and initialization) unless these routines are so large that good density cannot be maintained.
- Segment programs that have long running, well defined phases by function even if some code must be duplicated. This will improve working set stability.
- Remember that locality of reference applies only to the working set. The important consideration is to keep storage references confined to pages that would normally tend to be in main storage at the same time. It is logically immaterial whether the working set

consists of contiguous pages or of pages scattered throughout the program. However, experience has shown that internal fragmentation is reduced (and program readability is improved) if the working set consists of contiguous pages.

- Initialize each data area just prior to its use rather than at the beginning of the program. This will tend to prevent unnecessary paging activity. In addition, if a large area of virtual storage is reserved for handling a worst case situation, do not initialize it until the exact size needed is determined.
- High use data areas and I/O buffers may be grouped together in common storage using the COM assembler instruction. This allows convenient ways of aligning these areas on specific page boundaries if needed.
- Align large buffers on page boundaries. Buffer areas are “fixed” in main storage during I/O operations. Careless buffer alignment could cause additional pages to be needed. From a paging viewpoint, the optimal size for I/O buffers is the length of a page (or a multiple thereof).
- If possible, separate read only data areas from areas that will be changed. This could save page out operations since only those pages that are changed are rewritten to external storage. Bear in mind, however, that good locality is usually more important than strict separation of read-only and read-write areas.
- Order subroutines that are needed together or that are nested. For example, if the main program invokes subroutines A, B, and C, or if A calls B which in turn calls C, then place A, B, and C together.
- Consider segmenting very large arrays and data areas into page size units. Then process each segment instead of the entire array.
- Literals (including literal address constants) may be inserted into the same page where they are referenced. This can be accomplished by judicious use of LTORG statements that can be placed at the end of each functional routine. This also helps to attain “readability” of a program - something that is often non-trivial.
- In a multi-processor environment where multiple tasks may be running concurrently in the same job, any data that is accessed from more than one task should be serially referenced to avoid simultaneous access. Simultaneous access may yield incomplete changes.
- If SORT is called from the Assembler and Computer Center procedures are not being used, the following statement must be included for the SORT/MERGE program messages:

```
//stepname.SORTMSGS  DD  SYSOUT=A
```

7.5.2 Instruction Usage Hints

Here are some examples of instruction usage. Any contributions to this section by experienced programmers would be warmly welcomed.

- When RX type instructions are used for data not covered under a USING statement, an index register should **always** be specified even if it is not being used. If no indexing is being used, indicate its absence by coding a comma (,) or specify register 0 as the index register preceding the B2 specification. Omitting the comma causes the Assembler to assume the B2 specification is an X2 specification and the instruction is assembled with no base register assigned. Although the instruction will work, it requires more execution time because of the added cycles for indexing and can cause confusion when debugging.

  Example:

  `L  R4,0(,R6)` is faster than `L  R4,0(R6).`

- Branch on Count (BCTR)

  This instruction is generally used to decrement the contents of a register (R1) being used as a counter and branch to the address contained in R2 when the counter is non-zero. This instruction may be used to decrement the counter without branching by specifying register zero for the R2 field. This can be quite useful to prepare a register for the object of an EXECUTE instruction.

```
        LA    R5,L'FIELDA       GET LENGTH OF FIELDA
        BCTR  R5,0              GET MACHINE LENGTH
        EX    R5,MOVE           EXECUTE THE MOVE
        .
        .
        .
MOVE  MVC   FIELDB(1),FIELDA
```

- Load Address (LA)

  Normally this instruction is used to load a register with the address of some data or instruction. However, it may also be used to load a register with an absolute value between 0 and 4095, or to increment a register by an absolute value between 0 and 4095. Load address may also be used to set the high order byte of a register to zero.

  Example:

```
LA   R5,10           PUT ABSOLUTE VALUE 10 IN R5
   .
   .
   .
LA   R5,10(0,R5)     INCREMENT R5 BY 10
```

The load address instruction is faster than the load instruction and therefore should be used instead of

```
L     R5,=F'10'
    .
    .
    .
A     R5,=H'5'
```

To clear the high order byte of a register code

```
LA    R7,0(0,R7)      INCREMENT R7 BY ZERO AND SET
                      BITS 0-7 TO ZERO
```

- Shifting Instructions

  To multiply or divide by a power of 2, use a shift instruction instead of the multiply or divide instruction. For example, to divide a number in register 4 by 8, code

```
SRL   R4,3            DIVIDE BY 8
```

  The number of shift positions equals the power of 2 by which the register is multiplied or divided.

- SAVE macro

  The SAVE macro may be coded with an identifier name of up to 70 characters. When using subroutines, it is a good idea to use this form of the SAVE macro so that each subroutine may be easily located in a dump by using the EBCDIC translation at the right side of the dump. Version number and date may be coded with the CSECT name as part of this identifier, as illustrated below:

```
SAVE (14,12),T,SORTER-V7-4OCT88
```

- Return codes

  Each Assembler Language main program or subroutine should set the return code in register 15 prior to returning control to the operating system or calling program. The RC parameter of the RETURN macro may be used. If register 15 is not properly set, a garbage condition code may appear for the job step.

- ANDs and ORs

AND (N,NR,NI,NC)	AND-ing two bits results in 1 if both bits are 1; otherwise the result is 0.

OR (O,OR,OI,OC)	OR-ing two bits results in 1 if either bit is 1; otherwise, the result is 0.
Exclusive-OR (X,XR,XI,XC)	Exclusive-OR-ing two bits results in 1 when 1, but not both of the bits is 1; otherwise, the result is 0.
To set a bit to 0	Use AND. The mask should be all ones except for the bit position(s) to be set to 0, which alone should be 0.
To set a bit to 1	Use OR. The mask should be all zeros except for the bit position(s) to be set to 1, which alone should be 1.
To invert a bit	Use Exclusive-OR. The mask should be all zeros except for the bit position(s) to be inverted, which alone should be 1.
To set a whole field or register to zero	Use Exclusive-OR. Any field (or register) exclusive-OR-ed with itself causes the field (or register) to be set to zero.
To exchange the contents of two fields or two registers	Use Exclusive-OR (XC or XR).

The exclusive-OR instructions may be used to exchange the contents of two registers or two areas of storage without the use of a third register or a third storage area. The sequence

```
A  exclusive-ored  B
B  exclusive-ored  A
A  exclusive-ored  B
```

will swap two areas. Example:

```
XC       FIELDA,FIELDB
XC       FIELDB,FIELDA
XC       FIELDA,FIELDB
         .
         .
         .
XR       R4,R6
XR       R6,R4
XR       R4,R6
```

will swap the contents of FIELDA and FIELDB and will exchange the contents of registers 4 and 6.

7.6 Data Formats for Inter-Language Communication

The following tables 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 1<=n<=32767	CHARACTER*n 1<=n<=3267	CHAR(n) 1<=n<=32767	Length = n bytes

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

Figure 2. 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) Range: -9999 to 9999	INTEGER*2 Range: -32768 to 32767	FIXED BIN (1-15,0) Range: -32768 to 32767	Halfword Length = 2 bytes.
PIC S9(5-9) COMP (or COMP-4) Range: -(9)9s to +(9)9s	INTEGER*4 Range: -2147483648 to 2147483647	FIXED BIN 16-31,0) Range: -2147483648 to 2147483647	Fullword Length =4 bytes.
PIC S9(10-18) COMP (or COMP-4) Range: -(18)9s to +(18)9s	-----	-----	Two-word Length = 8 bytes.

Figure 3. Fixed Point Formats for Inter-Language Communication

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

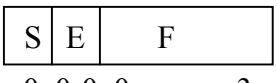
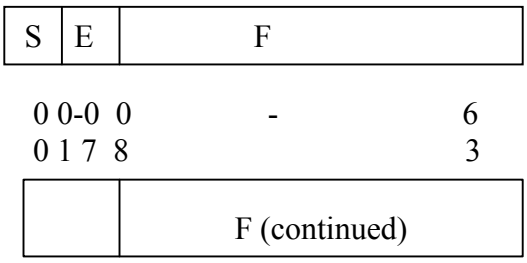
"S" is a binary sign bit: 0 is positive; 1 is negative.
"I" is a 15, 31, or 63 bit integer.
Figure 3 (Continued)

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 Magnitude: 10**-78 to 10**75 Precision: 7.2 digits	REAL*4 Magnitude: 10**-78 to 10**75 Precision: 7.2 digits	FLOAT DEC(1-6) Magnitude: 10**-78 to 10**75 Precision: 6 digits	Short Length = 4 bytes
COMP-2 Magnitude: 10**-78 to 10**75 Precision: 16 digits	REAL*8 Magnitude: 10**-78 to 10**75 Precision: 16.8 digits	FLOAT DEC(7-16) Magnitude: 10**-78 to 10**75 Precision: 16 digits	Long Length = 8 bytes
------	REAL*16 Magnitude: 10**-78 to 10**75 Precision: 35 digits	FLOAT DEC(17-33) Magnitude: 10**-78 to 10**75 Precision: 33 digits	Extended Length = 16 bytes

Figure 4. Floating Point Formats for Inter-Language Communication

MACHINE DATA FORMAT	EXAMPLES	
 Short	Value --- +1234 --- 1234	Internal hex representation --- 434D2000 --- C34D2000
 Long	+1234 --- -1234	434D20…0 --- C34D20…0
 Extended	+1234 --- -1234	434D20…0 --- 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 4 (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) DISPLAY 1<=n<=18 Range: 0 to (18)9s	------	PIC '(n)9' 1<=n<=15 Range: 0 to (15)9s	Unsigned Length = n bytes.
PIC S9(n) DISPLAY 1<=n<=18 Range: -(18)9s to +(18)9s	------	PIC '(n-1)9T' 1<=n<=15 Range: -(15)9s to +(15)9s	Signed Length = n bytes.

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

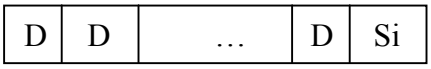
"Z" is a 4 bit zone code with a value of hex F.
"D" is a 4 bit binary decimal number with a value between hex 0 and 9.
"Si" is a 4 bit sign code: A, C, E, and F are positive; B and D are negative.

Figure 5. 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 PIC 9(n) 1<=n<=18 Range: -(18)9s to +(18)9s	------	FIXED DEC(n) 1<=n<=15 Range: -(15)9s to +(15)9s	Length in bytes = (n+1)/2 rounded up.

MACHINE DATA FORMAT	EXAMPLES	
 0-0 0-0 0 3 4 7	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 6. Packed Decimal Formats for Inter-Language Communication

INDEX

ABENDs
  B37 from Assembler Language, 17
  D37 from Assembler Language, 17
ASMHCALL procedure, 23
ASMHCOMP procedure, 9
ASMHLDGO procedure, 15
ASMHLKGO procedure, 10, 13, 25
ASMHLKMM procedure, 20
ASMHLKSM procedure, 17
ASMHOBJ procedure, 12
assembling and running programs, 9
B37 ABEND, 17
Binder conventions, 27
CALL procedure, 23
character data, 33
charges
  reducing CPU time, 28
coding hints, 29
comments for programs, 26
compiler options
  changing, 8
  NIH Computer Center, 8
CPU time
  reducing, 28
D37 ABEND, 17
data formats, 33
data types
  character, 33
  fixed point, 33
  floating point, 35
  packed decimal, 38
  zoned decimal, 37
DD statements needed, 10
documentation. *See* publications
documenting programs, 26
efficiency of programs, 28
electronic mail
  submitting a PTR, 1
ENTER PTR, 1
ENTER PUBWARE, 5
ENTRY statement, 22
Federal Information Processing Standard. *See* FIPS
figures
  character formats for inter-language communication, 33
  fixed point formats for inter-language communication, 33
  floating point formats for inter-language communication, 35
  packed decimal formats for inter-language communication, 38
  SYSOUT DCB information for Assembler procedures, 4
  zoned decimal data formats for Inter-Language Communication, 37
FIPS standard
  non for Assembler, 2
fixed point data, 33
floating point, 35
High Level Assembler, 1
INCLUDE statement, 22, 25
instruction usage, 29
interlanguage communications, 33
introduction, 1
LDGO procedure, 15
level of support, 1
libraries
  private, 23
LKGO procedure, 10, 13, 25
LKMM procedure, 20
LKSM procedure, 17
load module storage, 17
load modules
  recreate periodically, 17
modular programming, 26
multi-member libraries, 20
OBJ procedure, 12
overriding procedures, 10
packed decimal, 38
PDS
  multi-member programs, 20
  single member for load module, 17
  use, 17
performance
  improving, 28

private libraries, 20
  load modules, 17
procedure functions, 2
procedure names, 2
program commenting, 26
programming guidelines, 28
programming tips, 26
PTR, 1
publications, 5
  Assembler Language, 27
  ordering, 5
  OS/VS Binder conventions, 27
PUBWARE, 5
registers, 27
  use, 27, 29
savings with load modules, 17
single member libraries, 17
software
  improving efficiency, 28
standards
  FIPS
    none for Assembler, 2
support, 1
SYSOUT record formats, 4
Technical Information Office, 5
World Wide Web
  PTR submission, 1
  publication ordering, 5
  publications online, 5
WYLBUR
  ENTER PUBWARE, 5
zoned decimal data, 37

Using Assembler Language 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:

manual revised: 9/98