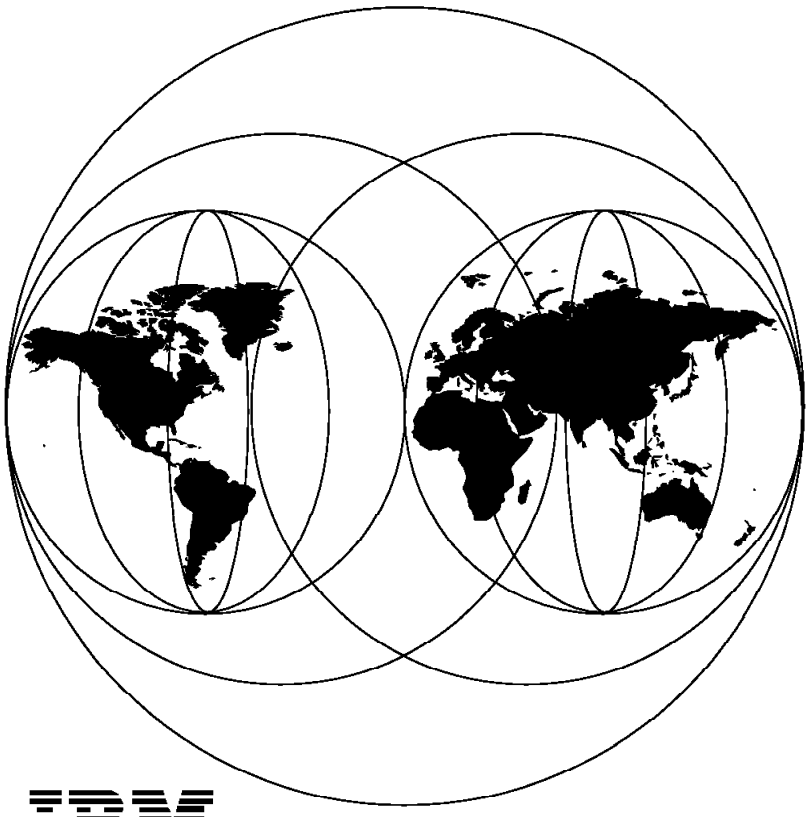
International Technical Support Organization SG24-3910-01

IBM High Level Assembler for MVS & VM & VSE Release 2 Presentation Guide

December 1995



IBM

International Technical Support Organization
San Jose Center

International Technical Support Organization

SG24-3910-01

IBM High Level Assembler for MVS & VM & VSE Release 2 Presentation Guide

December 1995

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page vii.

Second Edition (December 1995)

This edition applies to Version 1 Release 2 of IBM High Level Assembler for MVS & VM & VSE, Program Number 5696-234, for use with the MVS/ESA, VM/ESA, and VSE/ESA operating systems.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471/E2 Building 080
650 Harry Road
San Jose, California 95120-6099

Abstract

This document is unique in its comprehensive overview of the IBM High Level Assembler for MVS & VM & VSE (HLASM). It focuses on the management and organizational advantages of HLASM and provides helpful insights into the utility and power of many of its advanced features.

This document also provides information about HLASM's powerful new language features, many new options, enhanced diagnostic capabilities, greatly improved usability, tailorability for individual and workgroup use, and numerous interfaces for integration into advanced application development environments.

This document is written for those who design, code, debug, and maintain Assembler Language programs and their managers. In the "Technical Overview" presentation, familiarity with the IBM System/360, System/370, and System/390 Assembler Language will be helpful in understanding some of the examples.

(115 pages)

Contents

Special Notices

This publication will help programmers of the System/360, System/370, and System/390 Assembler Language and their managers understand the many benefits and features of High Level Assembler. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM High Level Assembler/MVS & VM & VSE, Release 1* and IBM High Level Assembler for MVS & VM & VSE, Release 2*. See the PUBLICATIONS section of the IBM Programming Announcements for HLASM Release 1 and HLASM Release 2 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

You can reproduce a page in this document as a transparency, if that page has the copyright notice on it. The copyright notice must appear on each page being reproduced. Foils should be reviewed in their entirety.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

DFSMS
DFSMS/MVS
ESA
Enterprise Systems Architecture/370
Enterprise Systems Architecture/390
High Level Assembler
HLASM
IBM High Level Assembler/MVS & VM & VSE
IBM High Level Assembler for MVS & VM & VSE
IBM
MVS/ESA
MVS/SP
MVS/XA
S/370
S/390
System/360
System/370
System/390
Virtual Machine/Enterprise Systems Architecture
Virtual Machine/Extended Architecture
VM/ESA
VM/XA
VSE/ESA

The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Other trademarks are trademarks of their respective companies.

Purpose of This Document

This document provides an overview of the many new features of IBM High Level Assembler for MVS & VM & VSE, a new high-function assembler for the IBM System/360, System/370, and System/390 series of machines.

How This Document Is Organized

This document is intended to provide Assembler Language programmers and their management with an introduction to the key features and benefits of IBM High Level Assembler for MVS & VM & VSE, Release 2. It contains two presentations, one emphasizing the organizational and managerial benefits of High Level Assembler, and the other providing technical information about the major enhancements supported by High Level Assembler.

The two presentations are:

1. A “Management Overview” presentation (with each foil prefixed with the letters MGMT) summarizes the managerial and organizational benefits of IBM High Level Assembler for MVS & VM & VSE.

   Note to the Presenter

   This presentation is nontechnical and should take about one-half hour. Additional time may be allotted for questions.

2. A “Technical Overview” presentation (with each foil prefixed with the letters TECH) provides an overview of the key technical features of IBM High Level Assembler for MVS & VM & VSE, examples of their use, and some indications of their benefits.

   Note to the Presenter

   This presentation is technical and should take about one hour. It assumes some familiarity with the Assembler Language, but detailed knowledge of or experience with the language is not required for most portions of the presentation.

   Additional time may be allotted for questions. Because some of the topics presented here may lead to further discussion, it may be appropriate to allocate from 60 to 90 minutes for full coverage of these topics.

The presentation text in the first two chapters may be suitable for use as handout materials. The Presentation Foils in Chapters 3 and 4 are identical to the mini-foils in the first two chapters.

Note to the Presenter

The two presentations have similar structure, but with a different emphasis. There is a fairly small overlap between the two presentations, so that you may want to select portions of each to suit the needs of your audience. You will probably not want to give your presentation by simply following the full text of the presentation materials, because more information is provided in many places than you will typically require.

Related Publications

The publications listed in this section are considered particularly suitable for more detailed discussions of the topics covered in this document.

- HLASM Release 2, MVS & VM Edition
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 General Information, MVS & VM Edition*, GC26-4943-01
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Licensed Program Specifications*, GC26-4944-02
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Language Reference, MVS & VM Edition*, SC26-4940-01
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Programmer's Guide, MVS & VM Edition*, SC26-4941-01
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Installation and Customization Guide, MVS & VM Edition*, SC26-3494-00

- HLASM Release 2, VSE Edition
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Licensed Program Specifications*, GC26-4944-02
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 General Information, VSE Edition*, GC26-8261-00
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Installation and Customization Guide, VSE Edition*, SC26-8263-00
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Programmer's Guide, VSE Edition*, SC26-8264-00
  - *IBM High Level Assembler for MVS & VM & VSE, Release 2 Language Reference, VSE Edition*, SC26-8265-00
- HLASM Release 1
  - *IBM High Level Assembler/MVS & VM & VSE, Release 1 Fact Sheet*, GC26-3189-00
  - *IBM High Level Assembler/MVS & VM & VSE, Release 1 Diagnosis Guide*, SC26-3110-00
  - *IBM High Level Assembler/MVS & VM & VSE, Release 1 Language Reference*, SC26-4940-00
  - *IBM High Level Assembler/MVS & VM & VSE, Release 1 Programmer's Guide*, SC26-4941-00
  - *IBM High Level Assembler/MVS & VM & VSE, Release 1 Installation*, SC26-4942-00

International Technical Support Organization Publications

The following publication describes HLASM Release 1:

- *IBM High Level Assembler/MVS & VM & VSE, Release 1 Presentation Guide*, GG24-3910-00, which will be replaced by this new document, SG24-3910-01.

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOPUB LISTALLX. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Visa and Master Cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser (such as WebExplorer from the OS/2 3.0 Warp BonusPak) to the following:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. Point your web browser to the IBM Redbooks home page:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Acknowledgments

This project was designed and managed by the author:

John R. Ehrman
IBM Software Solutions Division
Santa Teresa Laboratory
San Jose, California USA

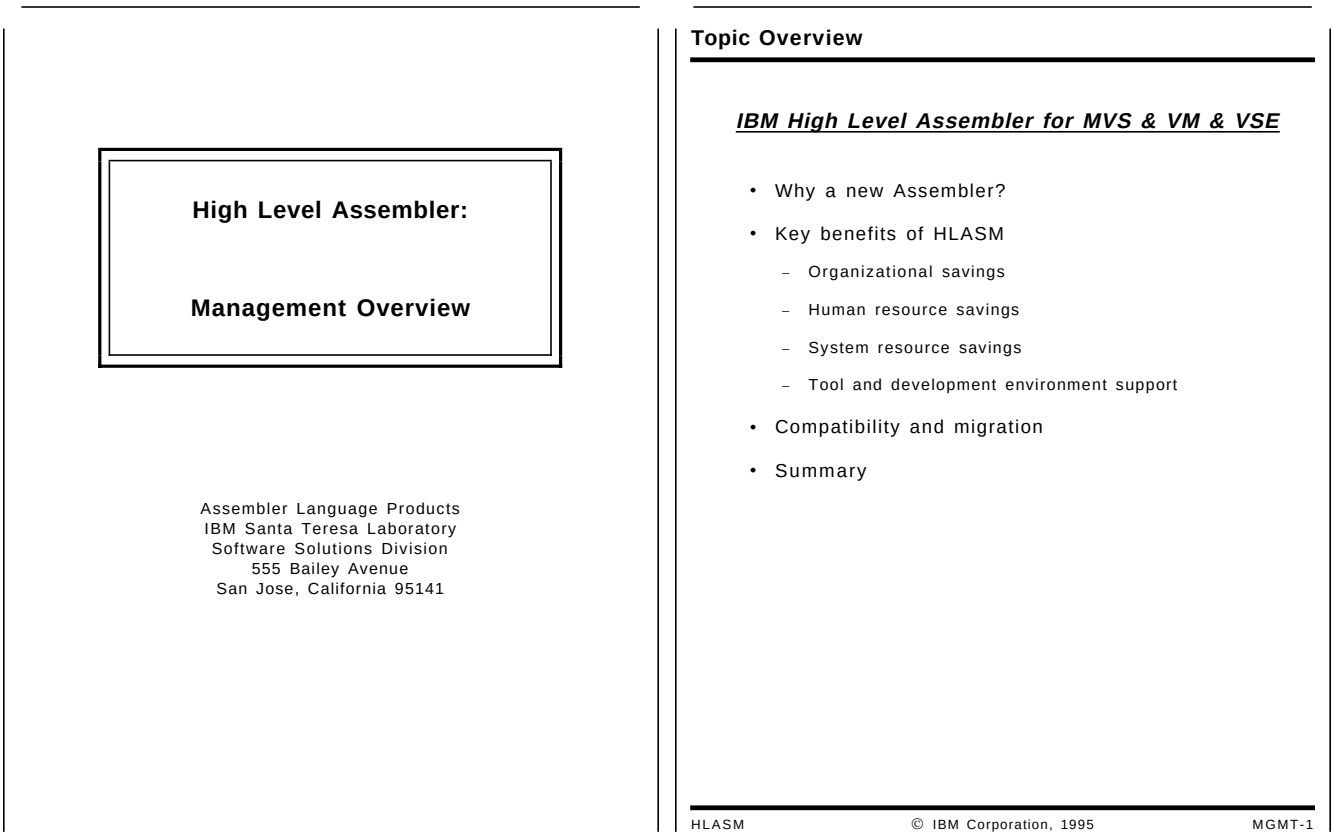
Thanks to:

Ueli Wahli
International Technical Support Organization, San Jose Center

Maggie Cutler
Editor

for the invaluable advice and guidance provided in the production of this document.

Chapter 1. High Level Assembler: Management Overview



1.1 Topic Overview

On May 5, 1992, IBM* announced IBM High Level Assembler/MVS & VM & VSE, Release 1* (program number 5696-234). It is also known by a shorter name, High Level Assembler*, and an acronym, HLASM*. It became generally available to tens of thousands of customers worldwide on June 26, 1992. On January 26, 1995, IBM announced IBM High Level Assembler for MVS & VM & VSE, Release 2. It became available on March 24, 1995.

HLASM has many features important to developers and maintainers of applications and application components written in Assembler Language. These features provide savings in human and system resources as well as interfaces for enabling integration into a wide variety of tools and environments.

HLASM is a complete replacement for the current IBM Assembler H Version 2 (program number 5668-962, also known as HASM, ASMH, or the “IEV Assembler”), Assembler XF (program numbers 5741-SC103, 5749-SC103, and 5752-SC103, also known as the “IFOX Assembler”), and the DOS/VSE (program number 5745-SC-ASM) and VSE/AF assemblers (program numbers 5686-066-14 and 5686-032, also known as the “IPK Assembler”).

This presentation provides a managerial (nontechnical) overview of the value and benefits of HLASM. A technical overview and detailed examination of the new features are provided in Chapter 2, “High Level Assembler: Technical Overview” on page 17.

1.2 Why a New Assembler?

IBM has recognized the continuing importance to its customers of Assembler Language applications and their underlying Assembler support.

- IBM′s System/370 and System/390 “mainframe” customers have made substantial investments in assembler-based applications and application components.

  Customer surveys and discussions have shown that these assembler-based applications, which frequently are critical to the business of the enterprise, are still under maintenance and development.

  Among the reasons for continued investment in Assembler Language applications are:

  – Assembler Language is the most flexible, adaptable, and universal language on the System/360, System/370, and System/390 platforms. It provides access to all permitted system services and interfaces easily with any other language.

  – Assembler Language provides complete control over instruction sequences (without depending on the capabilities or peculiarities of a compiler) while giving maximum performance.

  – Assembler Language can be written to be highly portable across System/360, System/370, and System/390 systems.

  - Using simple macro techniques, Assembler Language applications can be made easily portable to other platforms, and developers can build their own application-specific language elements in easily extended increments.

- HLASM helps protect customer investments in existing Assembler Language applications by extending their productive lifetimes.

  The new capabilities of HLASM can provide immediate benefits, with no “learning curve.” With a small investment, the other new features can be exploited to provide a wide variety of advantages.

- People with the skills required to support Assembler Language applications are becoming harder to find, train, and/or retain, and they are usually a critical resource for their companies. It is important to maximize their productivity.

  Every increment in efficiency provided for assembler programmers helps to “leverage” their increasingly rare and valuable skills; HLASM provides extensive enhancements in areas that promote greater usability, efficiency, reliability, and productivity.

  HLASM contains many enhancements specifically targeted to satisfying the needs of developers and maintainers of all Assembler Language applications, especially medium- to large-sized applications.

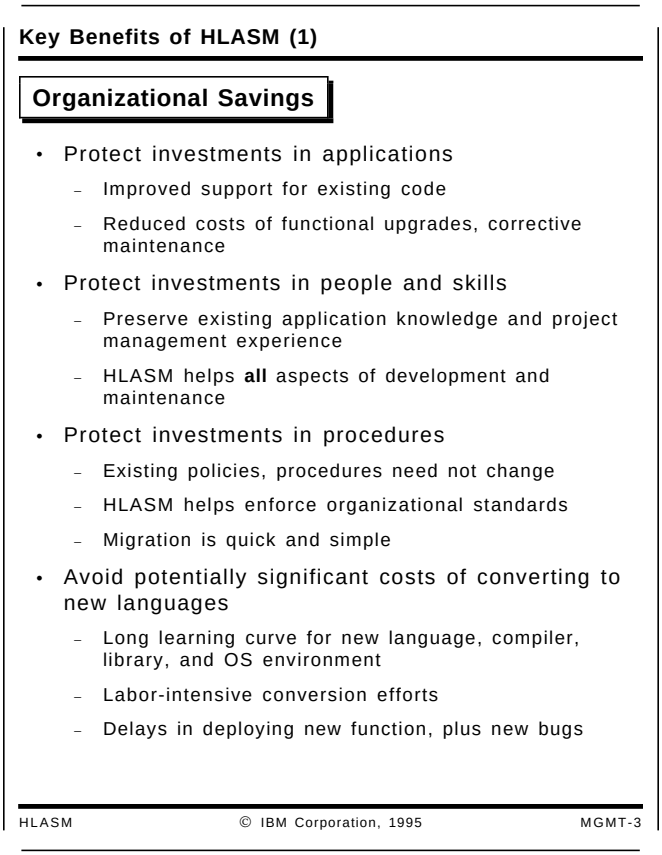
- There has been a steady stream of user requirements for enhancements and improvements to the IBM assemblers.

  HLASM satisfies a very large number of customer requests and user-group requirements for assembler improvements.

- The existing DOS/VSE Assembler and Assembler H were designed for the small real-storage environments of the late 1960s; the fact that they have served so well for so long is a remarkable tribute to the strength of their designs. However, the needs of assembler programmers have changed and evolved over the nearly 30 years since Assembler H and the DOS/VSE Assembler were introduced: today′s applications are larger, more complex, and harder to manage.

  - HLASM integrates almost all functions of past Assemblers into a single product that runs on MVS/ESA, VM/ESA, and VSE/ESA. Thus application developers no longer need to worry about the differences among Assembler products.

  - In particular, the needs of the VSE/ESA customer for new function, performance, and usability have been addressed in a single high-function Assembler that supports all of IBM′s mainline System/370 and System/390 operating systems.



1.3 Key Benefits of HLASM (1)

HLASM provides numerous benefits to you and your organization by protecting your investments in applications, people, and procedures.

- **Protect investments in applications**

  By providing improved technology to support existing applications, HLASM can extend the useful lifetimes of those applications while reducing the costs of enhancement and maintenance.

- **Protect investments in people and skills**

  The knowledge and skills of an organization′s application programmers are a valuable resource. HLASM helps to maximize the productivity of your application programmers by relieving them of many tedious and unproductive tasks that it can now do itself.

- **Protect investments in procedures**

  It takes many years to develop efficient and reliable project management procedures such as estimation, tracking, and analysis. HLASM can extend the useful lifetime of these procedures by making more efficient use of the human and system resources.

- **Avoid potentially significant costs of converting to new languages**

  HLASM helps you avoid converting existing—and working—applications from Assembler Language to other languages. Not only do such conversions

delay the introduction of new function into applications, but they also typically require a long “learning curve” for the new language, its compiler and run-time behaviors, and its interactions with the operating system environment. In addition, conversions tend to introduce new bugs into code that previously ran correctly.

Note to the Presenter

You can use the information below for background and discussion material if your audience wants to discuss factors involved in converting to a new programming language.

A recent research paper helps in understanding some typical motivations for, and results of, converting to new languages.[1] Research at 16 organizations showed that the reasons *for* changing computer languages included:

- *Standardization* on a smaller number of languages
- *Portability* across multiple platforms
- *Industrial strength* languages with greater power
- *Structured Query Language* support
- *Bulk buying discounts*.

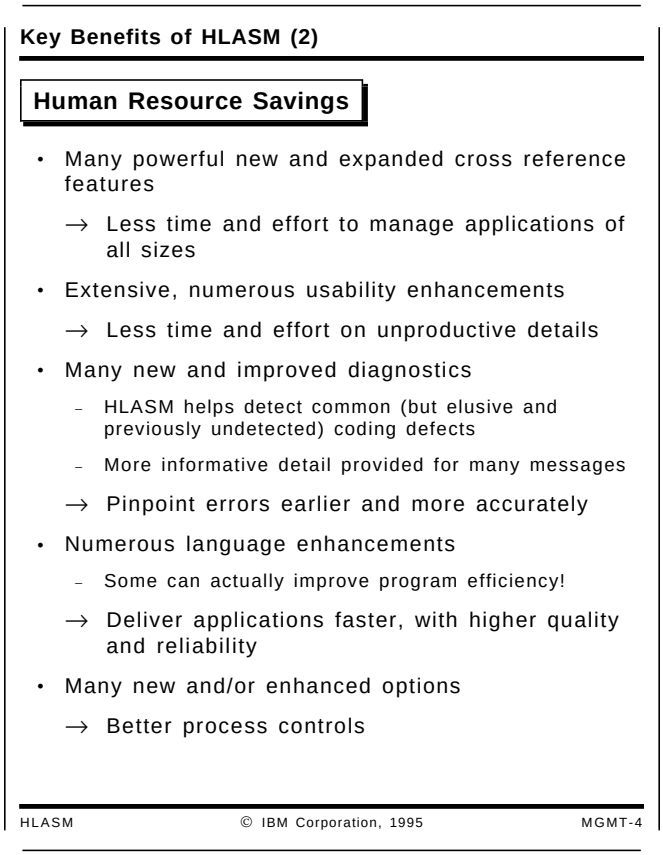
The reasons for *not* changing computer language included:

- *Hardware cost* due to increased overheads of the new language
- *Language capability and reliability* uncertainties
- *Continuity* loss due to new development methodologies, training costs, and invalidated project estimation experience
- *Language translator cost*, including run-time libraries.

The conclusions of the research were:

1. "(T)he key drivers of productivity are social and not technical."
    - *Standardization:* "(T)he key motivation for changing computer language was standardization. The use of only one language was expected to improve staff productivity.... While this is a reasonable assumption it is simplistic.... The goal of standardization may not be a practical one."
    - *Long term productivity:* "The analysis ... shows that program writing takes just 13% of a programmer's time."
2. "The costs of changing languages were higher than expected" and include delays due to the need for additional hardware acquisitions.
3. "(C)hanging languages hinders estimating, planning, and project management."
    - *People costs:* "The costs in staff time were high and often hidden because of the need to alter the development life cycle.... It was difficult for them to adopt a new life cycle. By rendering their experience invalid, both estimating and project management were made more hazardous. The costs of this were considerable."
    - "(T)he key drivers for organization performance are sociological not technical. Therefore the technocratic response of changing the computer language is not appropriate."
4. The overall conclusion was: "Changing languages should be avoided if possible."

[1] Peter J. Middleton, The Costs of Changing to a Fourth Generation Computer Language, *Journal of Programming Languages*, **2**, (1994) 67-76.



1.4 Key Benefits of HLASM (2)

HLASM provides many extensions to past IBM Assemblers. These extensions enable substantial savings in time and human and machine resources and support integration of HLASM into tool and development environments.

1.4.1 Human Resource Savings

Numerous enhancements in HLASM reduce the time and effort required to develop and maintain Assembler Language applications.

Note to the Presenter

For each item on the foil, additional detail is provided in the sublists below. Use this detail if your audience wants specific examples of particular benefits. (These items are described in greater detail in Chapter 2, “High Level Assembler: Technical Overview” on page 17.)

- Many new and expanded cross reference features are provided. These can significantly shorten the time required to accomplish many of the tasks involved in maintaining and managing applications.

  The features include:

  – A catalog and cross reference of macro and COPY-member usage, source-file record origins, and additional information in the summary

statistics allow HLASM to track the precise file or data set origin of *every* input record used in the assembly.

Note: This feature alone can save hours of expensive and tedious detective work in searching for the causes of versioning problems!

- A summary of all DSECT definition-start statements and their lengths
- A USING map summarizing all USING- and DROP-statement activity
- Many symbol-XREF extensions, including tags for modification, branch, and EXecute targets, and for appearance in USING and DROP statements.

  Note: This feature eliminates the tedium in hunting for the few instructions that might have changed the value of a variable, or for variables that might be part of an addressing expression.

- Many large and small usability enhancements improve the ease with which applications can be developed and maintained.

  The enhancements include:

  - Extensive PRINT and listing controls
  - Mixed-case input
  - USINGs-in-effect headings on each page

    Note: This enhancement greatly improves the understandability of the code on each page of the program listing.

  - Blank lines for spacing
  - Improved terminal output.

- HLASM provides many enhanced and new diagnostics. These help speed the processes of locating and correcting errors as well as avoid other possible sources of error. They also reduce the technical burdens on individuals with Assembler Language expertise.

  The enhanced and new diagnostics include:

  - Warnings are now issued for common (but previously undetected) USING and statement-continuation errors, thus reducing the time needed to produce a correct program.
  - More information is provided for many existing messages. For example, every diagnostic may be accompanied by a second message identifying the file and ordinal record number from which the flagged statement was taken.
  - Messages may be requested in either uppercase or mixed-case English, or in German, Spanish, or Japanese.
  - Program information in messages appears in a grammatically sensible position, and the messages have been made more informative. For example, the “Undefined Symbol” message includes the symbol in question.

- HLASM provides numerous language enhancements that can materially improve the speed and accuracy of application development and the quality and reliability of the resulting code.

The language enhancements include:

- Three new USING statements, which improve program reliability and efficiency and let you manage complex data structures far more simply, naturally, and effectively than was possible with previous Assemblers.

  Note: Surprising as it may seem, these new USING statements allow you to write *more efficient code* than you can today! (The details in the "Technical Overview" presentation are described in 2.9, "New and Enhanced Base Language (1)" on page 40 and 2.11, "Examples of New USING Statements and Diagnostics" on page 45.)

- Many new system variable symbols, which provide added powerful function and richer access to properties of the assembly-time environment, to facilitate the tailoring of applications to specific requirements. They also let you easily capture useful information into the object code.

• Existing assembly-time options have been extended and enhanced, and many new options have been added. These options allow increased flexibility and precision in controlling the processes you use to manage application development.

  These assembly-time option enhancements include:

  - The USING option provides controls over several different warning conditions. This option helps programmers diagnose and control one of the most complex and error-prone aspects of managing Assembler Language programs: USING statements and addressability controls.

  - The COMPAT option controls the treatment of areas where changes to the source program might cause HLASM to behave differently from previous Assemblers.

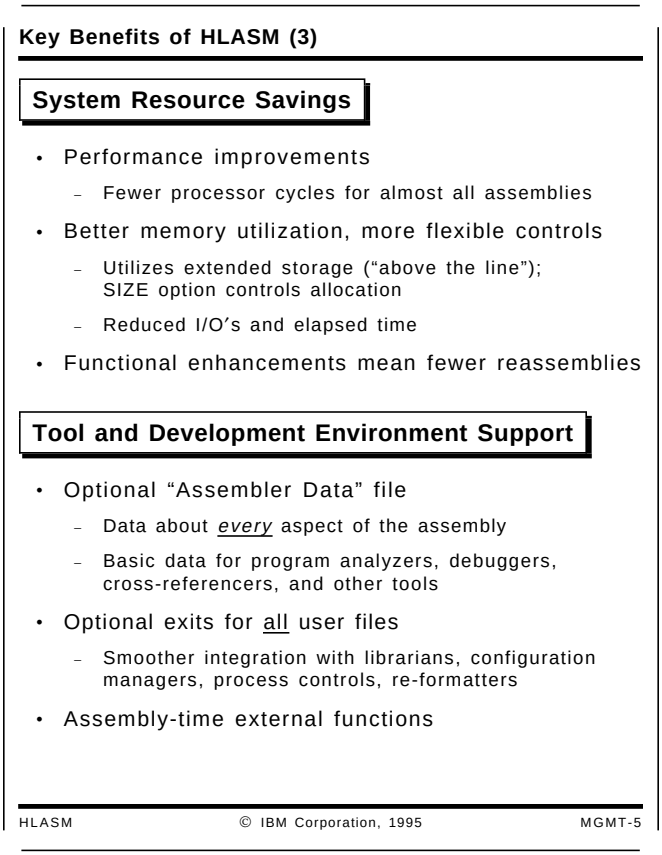
  - The PCONTROL option permits you to override source-program print controls, thus allowing you to produce a "full" and readable assembly listing for the entire program, without having to modify *any* source statements. The PCONTROL(MCALL) option and the PRINT MCALL statement allow you to direct the Assembler to produce information about the actual text of all inner macro calls.

  - Key elements of the assembly-time environment are loaded dynamically at Assembler invocation: the default options table, the translation table, the operation-code table, and the messages table. This enhancement permits simple install-time and invocation procedures for all Assembler users in an installation.

  - The TRANSLATE option simplifies the creation of applications that support national languages other than English.

  - The XOBJECT option enables Assembler support of new features of the DFSMS/MVS Binder, allowing you to use external names up to 63 characters long to create multimodal and multicomponent modules, and much more.

  To assist in ensuring that organizational standards are followed, you can install HLASM with selected options "fixed" so that they cannot be overridden when the Assembler is invoked.



1.5 Key Benefits of HLASM (3)

1.5.1 System Resource Savings

The performance of HLASM Release 2 has been improved over that of HLASM Release 1 in several areas: reducing I/O requirements and elapsed times to complete assemblies, and processor utilization (in almost all cases).

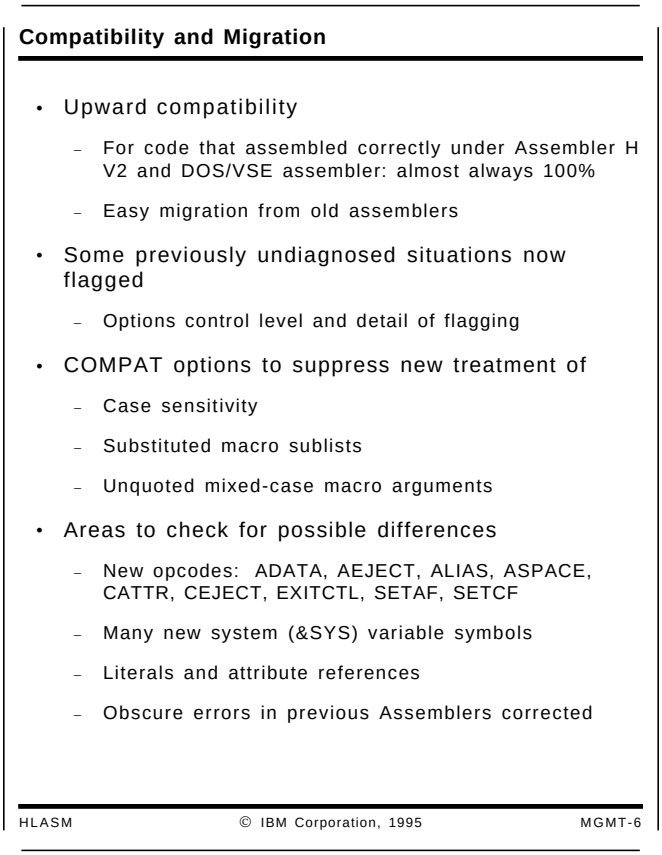
- Reduced processor utilization
  - Small assemblies are initiated and completed more quickly.
  - Large macro-bound assemblies (such as IMS system generation) require between 10% and 15% less processor time than HLASM Release 1.

  Overall, users will see relatively minor differences in processor utilization between High Level Assembler and Assembler H.
- Internal enhancements in HLASM are directed toward performance improvements such as better memory utilization (for example, working storage above 16MB and improved utility-file management) and controls (for example, the new SIZE option). HLASM′s ability to utilize storage above 16MB can reduce or eliminate the need for utility-file operations.
- Another significant benefit of the many improvements in HLASM is that fewer assembly steps and less elapsed time are required to produce a completed application.

1.5.2 Tool and Development Environment Support Facilities

HLASM provides facilities that support the integration of the Assembler into tool and development environments. The most important support features are:

- An optional Assembler Data file that contains data about *every* aspect of the assembly: all input records and their sources, all messages, all symbols, complete cross reference information, and much more. This file can be used as a basic source of data to support a great variety of tools such as analyzers and debuggers.
- Optional user exits are provided for inspecting and monitoring the flow of records to and from all user files. With this feature the Assembler can be integrated with librarians, configuration managers, and other components that can benefit from direct and immediate interaction with the Assembler's input and output streams.
- HLASM supports assembly-time external functions. These provide the application developer with essentially unlimited access to any capability of the assembly-time environment and support the programming of complex assembly-time computations and interactions that would be difficult or impossible to achieve without them.



1.6 Compatibility and Migration

A key design consideration for HLASM is that existing code should continue to assemble correctly, with the appropriate option settings.

Because the enormous existing base of assembler code must continue to be supported, HLASM provides as nearly full upward compatibility with previous Assemblers as possible.

Note to the Presenter

You may want to skip the details on this foil. However, if compatibility questions are important to your audience, some specific factors are listed here; more details are covered in the “Technical Overview” presentation.

Some possible compatibility differences are:

- The COMPAT option controls HLASM′s treatment of macro sublists and mixed-case input text. If you set the COMPAT option appropriately, HLASM behaves just as Assembler H (and, for cases not already documented as significant differences, the DOS/VSE Assembler) behaved.
- If you enable some of the new diagnostics, some conditions might be flagged that were not detected by the DOS/VSE Assembler or by Assembler H. Among these are USING-statement overlaps (and misuses) and common

continuation-statement errors. You can control this flagging by using appropriate option settings.

The areas to check for possible sources of incompatibility between HLASM and previous Assemblers are few:

- If you use macros with the names of the nine new opcodes:

  ADATA, AEJECT, ALIAS, ASPACE, CATTR, CEJECT, EXITCTL, SETAF, SETCF

  you might have to use OPSYNs or other bypasses to ensure that your macros are used instead of the new “native” opcodes.
- Many new system (&SYS) variable symbols have been introduced. If your program defines its own local or global variable symbols starting with the characters &SYS (it should not!), name collisions may occur.
- HLASM supports literals and symbol attribute references in wider contexts than previous Assemblers, so that certain complex or unusual constructions may be interpreted differently.
- A few minor (and very obscure) errors in the behavior of Assembler H have been corrected.

Each IBM High Level Assembler for MVS & VM & VSE *Programmer's Guide* provides a comprehensive list of differences between HLASM and its predecessors.

1.7 HLASM Summary

This presentation necessarily provides only a brief overview of the many new capabilities of High Level Assembler.

The usability, language, listing, and other enhancements embodied in HLASM make development and maintenance of Assembler Language applications easy and efficient. Thus the savings in people time alone justify the small effort required to make the transition from previous Assemblers to HLASM.

In summary, HLASM provides greatly enhanced Assembler technology to:

- Preserve investments in code, people, and processes
- Improve support for critical assembler-based applications
- Maximize the productivity of personnel with critical skills
- Support integration of the Assembler with modern application development tools and environments
- Help avoid costs of unnecessary conversion of existing applications to other languages.

The productivity gains obtained from the use of HLASM can be found in many areas:

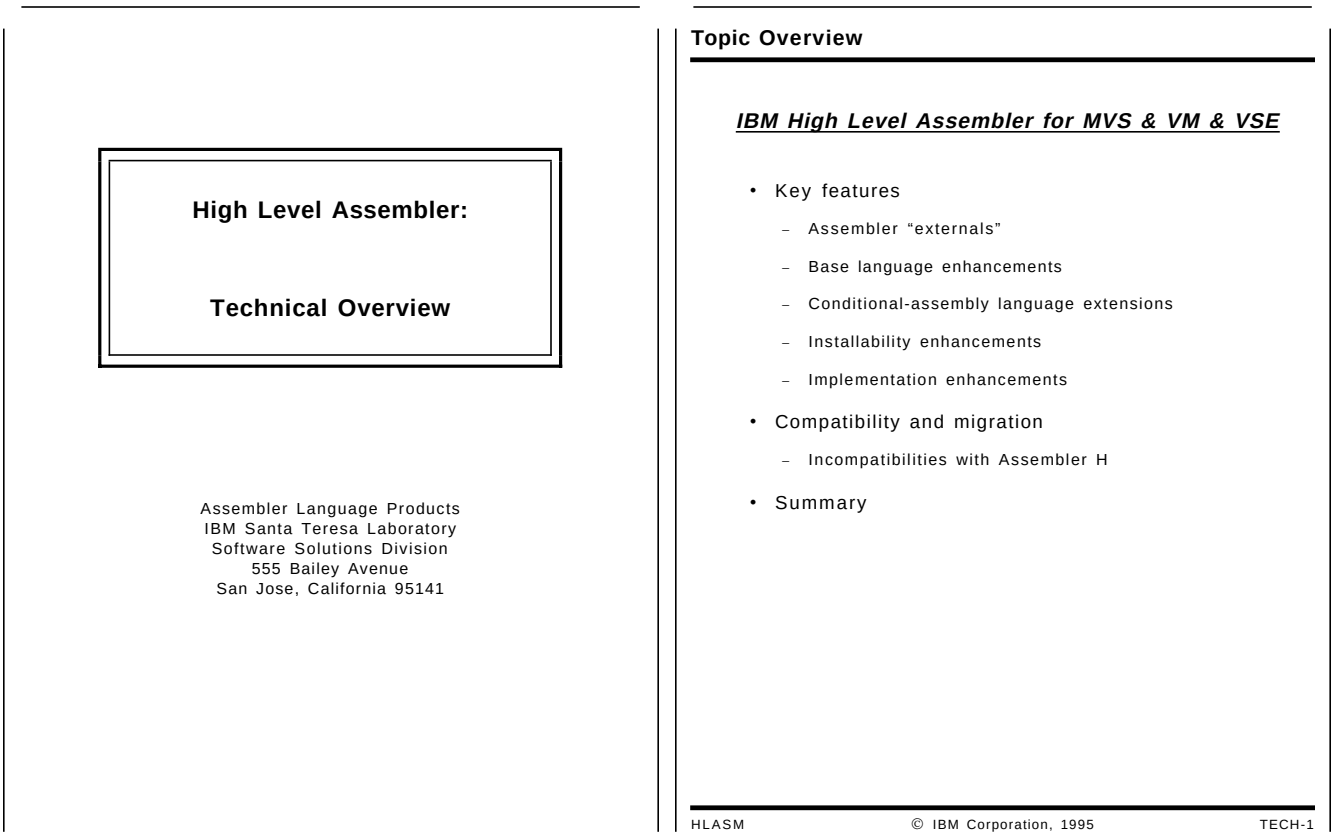
- Numerous enhancements to the base language, as well as to the inner conditional-assembly or macro language improve the readability, maintainability, and efficiency of Assembler Language applications.
- Extensive usability enhancements improve the efficiency and productivity of assembler programmers and maintainers.
- Many new and expanded diagnostic capabilities help in delivering higher-quality and more reliable applications.

Using HLASM will result in cost savings attributable to:

- Faster development cycles due to performance and usability improvements
- More maintainable and reliable applications due to new language, diagnostic, and extensive support features
- More efficient and productive programmers.

IBM has provided a significant tool that can help Assembler application developers with every aspect of developing, debugging, and maintaining Assembler Language applications on its “host” System/360, System/370, and System/390 systems.

Chapter 2. High Level Assembler: Technical Overview



2.1 Topic Overview

On May 5, 1992, IBM announced IBM High Level Assembler/MVS & VM & VSE, Release 1 (program number 5696-234). It is also known by a shorter name, High Level Assembler, and an acronym, HLASM. It became generally available to tens of thousands of customers worldwide on June 26, 1992. On January 26, 1995, IBM announced IBM High Level Assembler for MVS & VM & VSE, Release 2. It became available on March 24, 1995.

HLASM has many features important to developers and maintainers of applications and application components written in Assembler Language. These features provide savings in human and system resources as well as interfaces for enabling integration into a wide variety of tools and environments.

High Level Assembler is a complete replacement for the current IBM Assembler H Version 2 (program number 5668-962, also known as HASM, ASMH, or the "IEV Assembler"), Assembler XF (program numbers 5741-SC103, 5749-SC103, and 5752-SC103, also known as the "IFOX Assembler"), and the DOS/VSE (program number 5745-SC-ASM) and VSE/AF assemblers (program numbers 5686-066-14 and 5686-032, also known as the "IPK Assembler").

This presentation provides a technical overview of the value and benefits of the most important new features of High Level Assembler. A management overview describing the organizational benefits of HLASM is provided in Chapter 1, “High Level Assembler: Management Overview” on page 1.

IBM High Level Assembler/MVS & VM & VSE, Release 1 supports the following operating systems:

- MVS/SP V2 R2 (MVS/XA)
- MVS/SP V3 R1 (MVS/ESA)
- MVS/ESA SP V4 R1
- MVS/ESA SP V4 R2
- VM/XA SP 2 running CMS 5.5
- VM/XA SP 2.1 running CMS 5.6
- VM/ESA R1 (370 feature) running CMS 7. (**Note**: CMS 5 and CMS 6 are supported for migration; however, dependencies may preclude the running of these levels of CMS in some VM/ESA environments.)
- VM/ESA R1 (ESA feature) running CMS 7 and CMS 8. (**Note**: CMS 5, CMS 5.5, CMS 5.6, and CMS 6 are supported for migration; however, dependencies may preclude the running of these levels of CMS in some VM/ESA environments.)
- VSE/ESA V1 R2

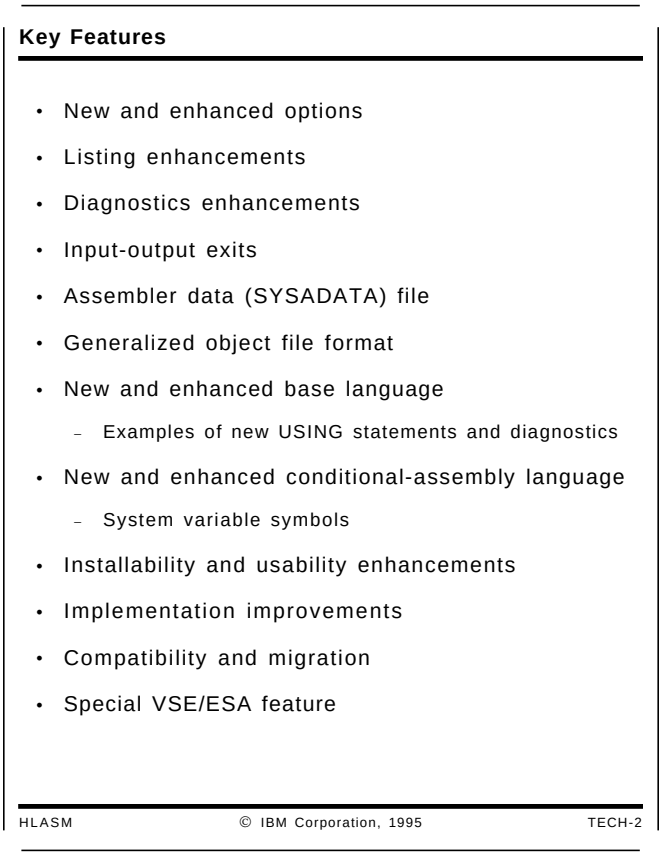
IBM High Level Assembler for MVS & VM & VSE, Release 2 supports the operating systems listed below. It is expected to operate under subsequent versions, releases, and modification levels of these systems:

- MVS/ESA SP V4 R2
- MVS/ESA SP V5 R1
- MVS/ESA SP V5 R2
  Note: Installation of High Level Assembler requires SMP/E.
- VM/ESA R1 (370 feature) running CMS 7
- VM/ESA R1 (ESA feature) running CMS 8
- VM/ESA R2 running CMS 9, CMS 10, or CMS 11
  Note: Installation of High Level Assembler requires VMSES/E and VMFPLC2.
- VSE/ESA V1 R2
- VSE/ESA V1 R3
- VSE/ESA V2 R1
  Note: Installation of High Level Assembler requires MSHP.

This presentation summarizes the technical aspects of the new features of HLASM in the following categories:

- Key assembler features, including enhancements to the Assembler Language and its display
- Major capabilities of High Level Assembler
- Enhancements to installability and Assembler internals.

Finally, compatibility and migration issues are discussed, particularly with respect to IBM Assembler H Version 2.1.



2.2 Key Features

High Level Assembler provides numerous new and improved features that assist developers and maintainers in many ways:

- New and enhanced options

  The new and enhanced options provide greater control over the operation of the Assembler. See 2.3, “New and Enhanced Options” on page 21 for details.

- Listing enhancements

  Extensive improvements have been made to the Assembler′s listing file, providing a great variety of useful information about the assembled program. These improvements are described in 2.4, “Listing Enhancements” on page 27.

- Diagnostics enhancements

  Error detection and diagnosis are greatly extended in HLASM. These diagnostics can help you to create a more robust, reliable, error-free, and maintainable program. The new diagnostic features are described in detail in 2.5, “Diagnostics Enhancements” on page 32.

- Input-output exits

  HLASM provides a full range of facilities for controlling all actions on its input and output files, through a very general and powerful set of

input-output exits. These are described in 2.6, “Input-Output Exits” on page 35.

- Assembler data (SYSADATA) file

  The SYSADATA (or ADATA) file provides complete information about every aspect of the assembly. It is described in greater detail in 2.7, “Assembler Data (SYSADATA) File” on page 37.

- Generalized object file format

  If requested by the XOBJECT option, HLASM Release 2 creates a new object file format. See 2.8, “Generalized Object File Format” on page 39.

- New and enhanced base and conditional-assembly languages

  The Assembler Language supported by High Level Assembler is actually two languages: the ordinary, or base, language in which machine instructions and data definitions are written, and the conditional-assembly language used to control and tailor statement sequences.

  Extensive enhancements have been made in many areas of both of these languages. See 2.9, “New and Enhanced Base Language (1)” on page 40 and 2.13, “New and Enhanced Conditional-Assembly Language (1)” on page 48.

- Installability and usability enhancements

  HLASM improves and simplifies the installation process, allowing greater controls over the content, options, and placement of the Assembler. See 2.16, “Installability and Usability Enhancements” on page 55.

- Implementation improvements

  Extensive internal improvements in HLASM increase its usability, reliability, and maintainability. See 2.17, “Implementation Improvements” on page 57.

- Compatibility and migration

  A design goal of HLASM is that it should continue to assemble code that assembled correctly under older assemblers. There are, however, some minor differences between HLASM and its predecessors; see 2.18, “Compatibility and Migration” on page 59 for details.

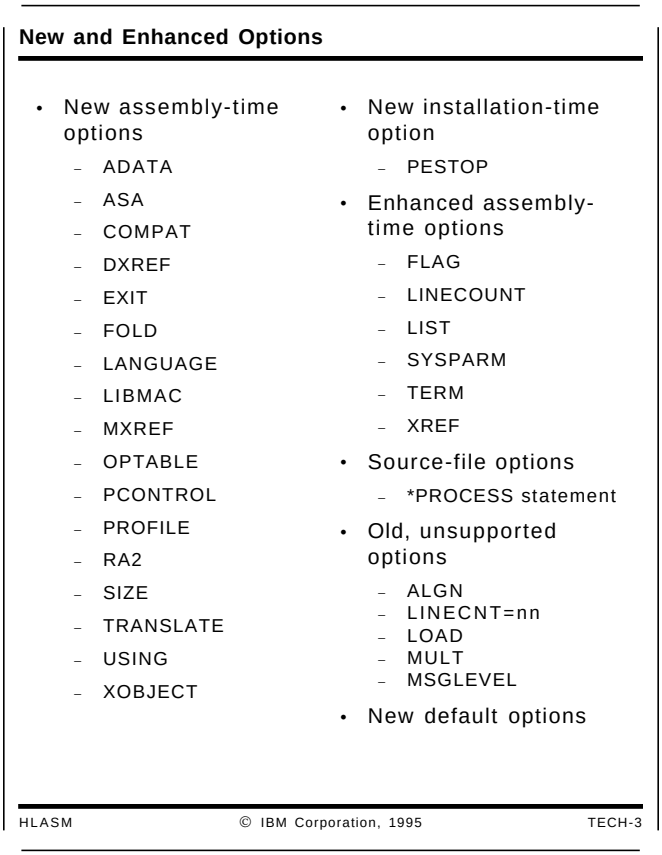
  Note to the Presenter

  If your audience does not use VSE/ESA, skip the following points; for a VSE/ESA audience, you should mention these items. No further operating system distinctions are made in this presentation.

- Special VSE/ESA feature
  - Greatly enhanced language capabilities
  - Support for XA and ESA instructions, and for generating object code capable of 31-bit addressing
  - Significantly improved CPU, I/O, and elapsed-time performance compared to the DOS/VSE Assembler.

  A specially tailored and lower-priced feature is provided for VSE/ESA systems. HLASM functions *not* available for VSE systems are:

  - Options: `ASA`, `LIST(133)`, `LIST(MAX)`, `XOBJECT`
  - Statements: `CATTR`
  - Sample exits: `INPUT`, `LISTING`, `ADATA`



2.3 New and Enhanced Options

Options handling and diagnosis have been improved in HLASM. Conflicting options are flagged with a low-severity warning, and attempts to override “fixed” options are disallowed. Many new options are supported, and extensions have been made to many existing options.

We describe HLASM options in these six areas:

- New assembly-time options
- New installation-time option
- Enhanced assembly-time options
- Source-file options
- Old, unsupported options
- New default options.

2.3.1 New Assembly-Time Options

HLASM provides 17 new options to help you control all aspects of the assembly process.

ADATA

Under control of the ADATA option, HLASM produces an Assembler data (SYSADATA) file containing useful and detailed information about every aspect of the assembly: its environment, all input and output files, the source program, all symbols and reference data, and the Assembler′s object

code output. This valuable and reliable information can be used to support tools for impact analysis, debugging, cross references, and a host of other housekeeping activities. Most important, it *completely eliminates* any need for listing scanners, system intrusions to trap supervisor calls, data-collection probes, and specialized local assembler modifications. See 2.7, “Assembler Data (SYSADATA) File” on page 37.

ASA

The ASA option lets you select either ASA (standard) or “machine” carriage-control spacing codes on the listing file. Thus you can integrate the assembler listing file with those of environments that use program librarians and similar configuration management tools.

COMPAT

The COMPAT option controls compatibility with Assembler H Version 2.1. It supports three suboptions: CASE, MACROCASE, and SYSLIST.

CASE

COMPAT(CASE) specifies that HLASM not allow lowercase characters in symbols and instruction mnemonics.

MACROCASE

HLASM allows symbols and other statement elements to be specified in mixed case. In some situations, you may want to write macro argument strings in mixed case, for macros that otherwise would be able to handle only uppercase argument strings. COMPAT(MACROCASE) specifies that HLASM should translate lowercase characters in unquoted macro arguments to uppercase before the macro is expanded.

SYSLIST

COMPAT(SYSLIST) instructs the Assembler to treat inner macro sublists as older assemblers treated them: as single character strings having no list structure.

DXREF

The DXREF option controls whether or not a DSECT cross reference should be printed in the output listing. The DSECT cross reference provides the starting statement number for every DSECT definition, its relocation ID, and its total length.

EXIT

The EXIT option allows you to specify exit routines to be invoked to monitor and control all I/O actions on any user file, including completely supplanting the Assembler′s actions. See 2.6, “Input-Output Exits” on page 35.

FOLD

The FOLD option causes lowercase characters to be folded to uppercase in the listing file. Such a facility is required in some environments where the Latin alphabet lowercase alphabetic code points are reserved for other uses (for example, Katakana).

LANGUAGE

The LANGUAGE option accepts one suboption specifying the language to be used for listing headings and diagnostic messages:

UE Uppercase English
EN Mixed-case English
DE German
ES Spanish
JP Japanese

LIBMAC

The LIBMAC option causes HLASM to treat library macros as though they were defined inline at the point of their first reference in the source program. With this option you can detect and track the causes of errors in library macro definitions without having to manually extract them from the library for insertion at a proper place into the source program.

MXREF(XREF), MXREF(SOURCE), MXREF(FULL)

The MXREF option specifies whether or not macro and COPY-member information should be included in the output listing. It has three suboptions: SOURCE, XREF, and FULL.

- The macro and COPY code source catalog specified by MXREF(SOURCE) precisely identifies the data sets from which every macro definition or COPY segment was retrieved. (Such information was not previously obtainable without extreme effort.)
- The cross reference specified by MXREF(XREF) provides data about the definition and each use of a macro or a COPY segment.
- MXREF(FULL) specifies that both the SOURCE catalog and the member XREF should be produced.

OPTABLE(instruction-set)

The OPTABLE option permits you specify which of several sets of machine instructions should be used for the assembly. This can also help to avoid creating code that cannot execute on the target machine. The allowed suboptions of OPTABLE are UNI, ESA, XA, 370, and DOS.

PCONTROL

The PCONTROL option can be used to override the internal settings of selected listing control (PRINT) statements appearing in the source program: [NO]DATA, [NO]GEN, [NO]MCALL, [NO]UHEAD, [NO]MSOURCE, OFF, and ON. With this option you can debug code that would otherwise be invisible or obscured in normal listings, without having to modify any element of the source code itself.

The PCONTROL(MCALL) option specifies that *all* macro calls should be displayed, not just top-level calls from open code. Macro calls are displayed in their original as-entered case, independent of the COMPAT(MACROCASE) option.

PROFILE(member)

The PROFILE option causes the Assembler to automatically COPY into the program all statements in the specified library member (or, by default, from ASMAPROF) before the rest of the input statements are processed. The statements included by PROFILE will follow any ICTL and *PROCESS statements already in the source stream.

RA2

The RA2 option controls whether or not two-byte relocatable address constants should be flagged. When the diagnostic is disabled, you can more easily use HLASM as a “cross-assembler” to create relocatable object code for smaller computers with 16-bit words.

SIZE

The SIZE option allows you to specify the amount of virtual storage to be allotted to the assembly, and whether or not it should allocate storage above 16MB.

TRANSLATE(table)

The TRANSLATE option lets you specify a translation table to be used to translate single-byte character data in character constants and literals to a different character set from the (default) EBCDIC. If you do not specify a table with the TRANSLATE option, HLASM uses an ASCII table.

This option can help you produce Assembler Language applications that support national language requirements.

USING

The USING option enables several new levels of diagnostic and listing information, to provide much finer control over common errors in the use of USING statements:

- The MAP suboption controls the appearance of the USING map.
- The WARN(nn) suboption controls several powerful diagnostics; see 2.5.2, “USING Warnings” on page 34.
- The LIMIT(xx) suboption checks for displacements exceeding the specified limit in addressability resolutions.

XOBJECT

Specifies that the object file should be written in the new extended object file format. (XOBJECT is exclusive with DECK and OBJECT.) See 2.8, “Generalized Object File Format” on page 39.

The ADATA suboption requests that Assembler Data (SYSADATA) be included in the extended object file.

2.3.2 New Installation-Time Option

One new installation-time option is provided.

PESTOP

The PESTOP installation-time option controls whether or not assembly-time option errors, or attempts to override options deleted at installation time, should cause the Assembler to stop or continue.

Previous assemblers diagnosed option errors, assigned defaults, and continued with the assembly. PESTOP requests that any error in the option list should terminate the assembly, thereby saving human and system resources by preventing wasted assemblies.

2.3.3 Enhanced Assembly-Time Options

Six existing options have been extended and enhanced.

FLAG

Four new suboptions are provided in the FLAG option. See 2.5.1, “New FLAG Suboptions” on page 33.

LINECOUNT

The LINECOUNT option can specify very long page lengths, or zero lengths to suppress automatic page skips. Specifying LINECOUNT(0) suppresses the effect of TITLE and EJECT, which permits creation of a vertically compacted listing.

LIST(121), LIST(133), LIST(MAX)

The LIST option lets you specify one of two different listing formats.

121 The Assembler produces the traditional and familiar listing of source statements with six-digit addresses

133 An expanded “wide” listing with eight-digit lengths and addresses and other useful information is provided

MAX Specifies that the Assembler should produce the widest possible listing consistent with the record length provided for the listing file when the Assembler was invoked.

SYSPARM

The maximum length of the &SYSPARM variable has been increased to 255 characters. Previously, it was limited to 8 characters; this extension permits greater flexibility in controlling the structure and content of Assembler Language programs.

TERM(WIDE),TERM(NARROW)

The terminal output display (on SYSTERM) has been simplified and improved; two new suboptions (WIDE and NARROW) may be specified. A new blank-compressed “narrow” layout enhances readability and allows the output to fit easily on an 80-character-wide display without wraparound.

- A single-line summary is given for a successful assembly. This summary line reduces the amount of clutter on your terminal when assembling a “batch” of modules.
- The Deck-ID (from a TITLE statement) is included in the summary message. This information helps you monitor the progress of a batch of assemblies and associate diagnostic messages with the proper portion of the input file.

XREF(UNREFS)

The ordinary symbol cross reference has been enhanced to provide a display of unreferenced symbols defined in nondummy control sections. This display is normally much shorter than a FULL cross reference, and can help in locating unneeded data or instructions. If XREF(FULL) is specified, HLASM ignores this option.

2.3.4 Source-File Options

You can use the *PROCESS statement to tailor assembly options for each source module. This source-file statement supplies option values for a single assembly. You may provide up to 10 *PROCESS statements; they must be the first statements in a source module, preceded only by an ICTL statement. Some options are not allowed on *PROCESS statements, and some are allowed only on the first assembly in a batch.

The hierarchy of options is:

1. Options specified at Assembler invocation time
2. Options specified on *PROCESS statements
3. Default options specified when HLASM Release 2 was installed, unless the option was specified as “fixed” at that time (in which case it cannot be overridden).

2.3.5 Old, Unsupported Options

Assembler H Version 2 supported some very old options (originally in the E-Level and F-Level Assemblers) that are not supported in HLASM.[2]

ALGN	This option must now be spelled ALIGN.
LINECNT=nn	This option must now be specified as LINECOUNT(nn).
LOAD	This option must now be specified as OBJECT.
MULT	This option must now be specified as BATCH.
MSGLEVEL=nn	This option must now be specified as FLAG(nn).

2.3.6 New Default Options

HLASM Release 2 provides different default option settings from both HLASM Release 1 and Assembler H. These changes are intended to improve the quality and reliability of Assembler Language applications. For details see Table 1 on page 60.

[2] In about 1970, IBM regularized option names so they would be more consistent across products. These old options were supported by Assembler H (which appeared in 1971) only for compatibility with its predecessor assemblers, none of which have been in use for many years.

Listing Enhancements

- General
  - Print-line records 121 characters or longer
  - Two listing formats: narrow (121), wide (133)
  - Headings controlled by LANGUAGE option
- Options summary page
  - Displays invocation, *PROCESS, and actual options; overriding ddnames
- External symbol dictionary
  - XOBJECT extensions, ALIAS information
- Source and object code listing
  - USINGs-in-effect heading lines
  - LOC, C-LOC, D-LOC, R-LOC location counter headings
  - USING resolution details: registers, offsets
  - 'C' statement-number tag for COPYed statements, '-' for AREAD
  - Location counter displayed in PRINT NOGEN regions
  - Improved page-break handling
- Relocation dictionary
  - Wide-format extensions

Listing Enhancements ...

- Ordinary symbol and literal XREF enhancements
  - Relocation ID, relocatability tags, symbol type
  - Compacted cross reference: leading zeros suppressed
  - Branch, Drop, Modification, Using, eXecute tags
  - XREF(UNREFS) for unreferenced non-DSECT symbols
- Macro and COPY-member summary and XREF
  - Data set ID, COPY and LIBMAC tags, where defined, who called
  - Cross reference shows all uses
- DSECT cross reference
  - Relocation ID, length, definition-start statement
- USING map
  - Statement-location data, base address, type of USING, anchor location or registers, last-resolved statement, USING text
- Diagnostic and assembly summary page
  - Pointers to origins of source statements
  - Assembler name and fix level
  - All files used, I/O and exit counts
  - Start and stop times, estimate of processor time

2.4 Listing Enhancements

The following listing enhancements will assist developers and maintainers in many ways:

- General
  - HLASM supports any print line length of at least 121 characters. These longer print-line records permit other tools (such as PRINT exits) to add information to the print line.
  - Listings may be requested in two forms, depending on the length specified for the print lines. The traditional format fits on a 121-character line; the new “wide” format fits on a line of at least 133 characters.

    The extended (or wide) *Source and Object Code* listing format is used if the record length specified for the listing (SYSPRINT) file is at least 133 characters, and this length is specified either explicitly or by default. While required for XOBJECT support, this capability is independent of the XOBJECT option.

    The Location Counter and ADDR1 and ADDR2 fields are expanded to eight digits, the statement-number field is expanded to six digits, and the full eight-character macro name appears in the sequence field of macro-generated statements.

  - Page and section headings are controlled by the LANGUAGE option and can therefore be in any of four languages, including uppercase or mixed-case English.
  - The page heading has been improved to provide dates including the century.
  - Address data for the *External Symbol Dictionary* (ESD), *Source and Object Code* listing, *Relocation Dictionary*, *Ordinary Symbol and Literal Cross Reference*, *DSECT Cross Reference*, *USING Map*, and other fields are displayed as eight hexadecimal digits in almost all situations, independent of the choice of wide or narrow listing format.

- Options summary page

  The first page of the listing file contains:

  - User-supplied options (from the invocation parameters)
  - *PROCESS options
  - List of all options in effect
  - List of overriding ddnames.

  Additional information is provided about errors in the requested options.

- External symbol dictionary
  - In addition to ALIAS information relating to the external symbol substitutions specified by ALIAS statements, the listing displays extra data when the XOBJECT option is active.

- Source and object code listing
  - The page heading can contain a summary of USINGs currently active as of the start of that page. Thus you can understand how symbolic addresses on each page will be resolved, without having to read the entire program up to that point.
  - A new location counter column heading indicates whether a CSECT, DSECT, RSECT, or COM section is currently in effect. The heading will be LOC, D-LOC, R-LOC, or C-LOC, respectively.
  - Much more detail is provided for USING-resolution displays. Both the second operand value and the registers specified as bases are shown for ordinary USINGs. The base-displacement resolution and first and second operand addresses used are provided for dependent USINGs. Dependent USINGs display the actual offset of the anchor location.
  - When PRINT NOGEN is in effect, HLASM displays the location counter value in effect for the first macro-generated code (if any) on the same line as the source statement. This makes debugging simpler for programs containing macro calls.
  - Records brought into the source program by COPY statements appear in the listing with a C character in column position “zero” (suffixed to the statement number), where a + character appears for macro-generated statements. Similarly, a − character appears for source statements read by AREAD statements.
  - HLASM provides improved page-break handling: it tries to eliminate blank pages caused by successive EJECT statements or by its forcing a page skip because a page was full (including conditions caused by SPACE statements) only to encounter an EJECT as the next statement.

Successive TITLE statements will cause no more than a single page skip, and only the last title text will be displayed.

  - If the FLAG(RECORD) option is specified, each diagnostic message will be followed by additional information about the statement such as its data set number, whether it is from the primary input stream or the library (and the member name if so), and the relative record number of the statement within its file. The post-assembly summary of flagged statements also includes this additional information.

- Relocation Dictionary
  - The Relocation Dictionary has been expanded to accommodate 31-bit addresses.
- Ordinary symbol and literal cross reference enhancements
  - The title of the symbol XREF has been changed to “Ordinary Symbol and Literal Cross Reference.”
  - The relocation ID (relocatability attribute) of each symbol is identified in the symbol XREF, which relates the symbol directly to its “owning” control section.
  - An additional column provides information about the relocatability properties of each symbol. Absolute symbols are flagged with an A, complexly relocatable symbols are flagged with a C, and simply relocatable symbols (the most common case) are not flagged.
  - The type attribute of each symbol is displayed.
  - Reference statement numbers have leading zeros and blanks suppressed, providing a more compact cross reference listing. That is, the references are free-form, separated by commas, rather than being in fixed columns. For cross reference and diagnostic summaries, most columnar statement-number entries are expanded to six positions.
  - In the cross reference, HLASM provides indicator tags for symbol usage in several contexts:
    - In USING (U) and DROP (D) statements
    - As targets of EXecute (X) instructions
    - Modification (M) tags for operand symbols naming fields whose contents may be modified by the action of the instruction
    - Branch target (B) tags for symbols used as operands of branch instructions.

    Note: The modification (M) tags permit rapid determination of which symbolic usages are for read references, and which are for write references. This feature eliminates much of the tedium in hunting for the few instructions that might have changed the value of a variable or the contents of a named register.
  - Symbols defined in ordinary (nondummy) control sections but not referenced elsewhere in the program may be selectively displayed by specifying the XREF(UNREFS) option, without the necessity of displaying *all* unreferenced symbols.

    Note: This can help in eliminating unneeded constants, storage areas, and “dead code” instructions.

- Macro and COPY member source summary and cross reference

  The MXREF option controls two parts of the macro and COPY member cross reference:

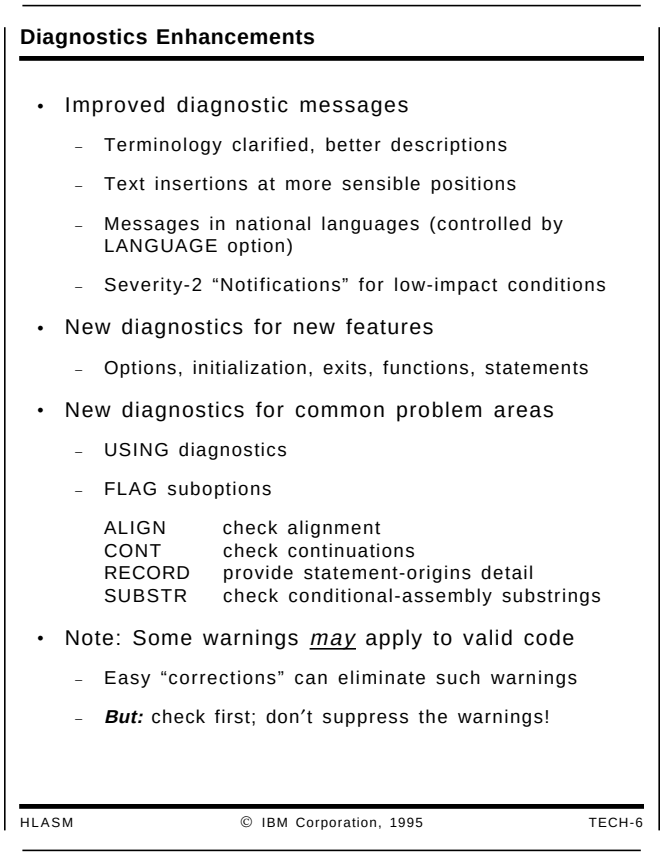
  - The source summary provides the data set or file name and volume identification for every file from which a member was taken, as well as an indication of whether it was a primary source (SYSIN) file or a library (SYSLIB) file, and a concatenation number to distinguish among concatenated input or library files. A list of members used is provided for each library file.
  - The cross reference provides, for each macro or COPY segment, its member name (if from a library), the concatenation number, which macro called it, the statement number at which it was defined, and the statement numbers at which references to it were made. (If MCALL is active, the information provided is even more accurate and detailed.)

  This information helps with version control, impact analysis, recompilation analysis, multiple library controls, and other code management tasks.

  Note: This feature alone can save hours of expensive and tedious detective work in searching for the causes of versioning problems!

- Dummy control section (DSECT) cross reference
  - The DSECT cross reference provides for each internal and external dummy section:
    - Section name
    - Relocation ID (this helps in identifying all symbols “belonging” to the section)
    - Section length
    - Statement number where the definition begins.
- USING map
  - The USING map summarizes all activity relating to the USING and DROP (and PUSH and POP USING) statements in the program that control resolutions of symbolic addresses into base-displacement form. The information provided includes:
    - Statement number of the USING or DROP
    - Section ID and location counter in effect at that point
    - Section ID and location counter specified as the base address of USING statements
    - All registers involved in USING or DROP actions
    - Maximum displacement calculated against each USING
    - Statement number for the last statement whose operand was resolved with respect to each USING statement (that is, the end of the USING’s range)
    - Text of the USING or DROP statement operands.
- Diagnostic and assembly summary page
  - All statements flagged are listed; if the FLAG(RECORD) option is active, each statement number is accompanied with information specifying the exact source record and the file from which it was read.
  - Expanded end-of-assembly summary information and statistics provide detailed information about the number of I/O actions, memory usage, number of diagnostic messages, host system, Assembler version, and other related data.

- All input and output data set names, member names, and volume IDs are displayed, listed by ddname.
- I/O exit statistics describe the exit type, exit name, number of calls to the exit, number of messages produced, and number of records added or deleted.
- The summary page includes the assembly start and stop times and an estimate of processor utilization.
- The final line of the summary page displays the return code for the assembly.



2.5 Diagnostics Enhancements

Run-time program failures are expensive and often difficult to debug. HLASM provides additional diagnostics to help detect programming errors earlier in the development cycle.

These features assist developers and maintainers in many ways:

- Improved diagnostic messages
  - The wording of all diagnostics has been clarified, regularized, and extended, and a severity-indicator suffix letter has been added to every message.
  - Many existing messages have been enhanced to provide more detailed information about the specific condition or object involved in the diagnosis.

    For example, the actual source-program variable or object that the Assembler considered to be the origin of the error condition is identified and inserted into the message text in a grammatically sensible place rather than always being at the end of the message.
  - You may request messages in four languages by setting the LANGUAGE option appropriately.

  - Some diagnostics for low-impact conditions are now treated as "Notifications," with severity code 2.

- New diagnostics for new features

  Many new diagnostics have been added in support of other new capabilities. For example:

  - Options-related messages
    - Attempt to override fixed option
    - Options conflicts
    - Options errors on *PROCESS statements
  - Assembler-invocation messages
    - Failure to load various functions, modules, and phases
    - Utility-file error conditions
    - File-open errors
  - Exit- and function-invocation messages
    - Exit-requested diagnostic messages, with six severity levels
    - Exit-requested or function-requested immediate termination
    - Invalid print-line length from exit
  - ALIAS-related messages
    - Multiple ALIASes for one external symbol
    - Illegal ALIAS
    - ALIAS not declared as external symbol
  - The CMS interface module, ASMAHL, provides numerous diagnostics for conditions specifically related to CMS.

- New diagnostics for common problem areas

  The intent of many of the new diagnostics is to warn about possible error conditions that previous assemblers ignored. Sometimes HLASM diagnoses code that is actually correct; but it is always better to verify such situations than to "automatically" suppress the warnings.

  - New FLAG diagnostics help to detect common oversights. They are described at 2.5.1, "New FLAG Suboptions."
  - New USING diagnostics are one of the most valuable features of High Level Assembler. They are described further at 2.5.2, "USING Warnings" on page 34.

Note: Occasionally, HLASM will flag correct code as possibly being incorrect. While it is easy to suppress the diagnostics, it is worth verifying that the code is indeed correct. It is usually very easy to modify the code to avoid the diagnostic, which is preferable to disabling HLASM′s checking entirely.

2.5.1 New FLAG Suboptions

The FLAG option has been greatly extended, with four new suboptions:

- FLAG(NOALIGN) causes HLASM to suppress the ASMA033W warning message when an alignment inconsistency is detected between a storage operand and a referencing instruction.
- FLAG(CONT) helps you to find one of the most insidious types of Assembler Language errors: when statements are continued, misplacement of a single character can cause portions of statements to be ignored. Specifying this

option causes HLASM to flag unusual or suspicious but difficult-to-find errors in specifying continued and continuation statements:

  - An operand on a continued record ends with a comma, and a continuation statement is present, but the continuing statement does not begin in the “continue” column (usually, 16).
  - A list of operands ends with a comma, but the continuation column (usually, 72) is blank.
  - The continuing statement starts in the continue column, but there is no comma present following the operands on the previous continued record.
  - The continued record is full, but the continuation record does not start in the continue column.

- The FLAG(RECORD) option causes HLASM to provide supplementary information (with each diagnostic message, and in the diagnostic summary) about the data set name and relative record number for the statement involved. This option can help with locating the specific original source statement requiring correction.
- FLAG(SUBSTR) specifies that HLASM should diagnose improper character substrings in the conditional-assembly language. For example:

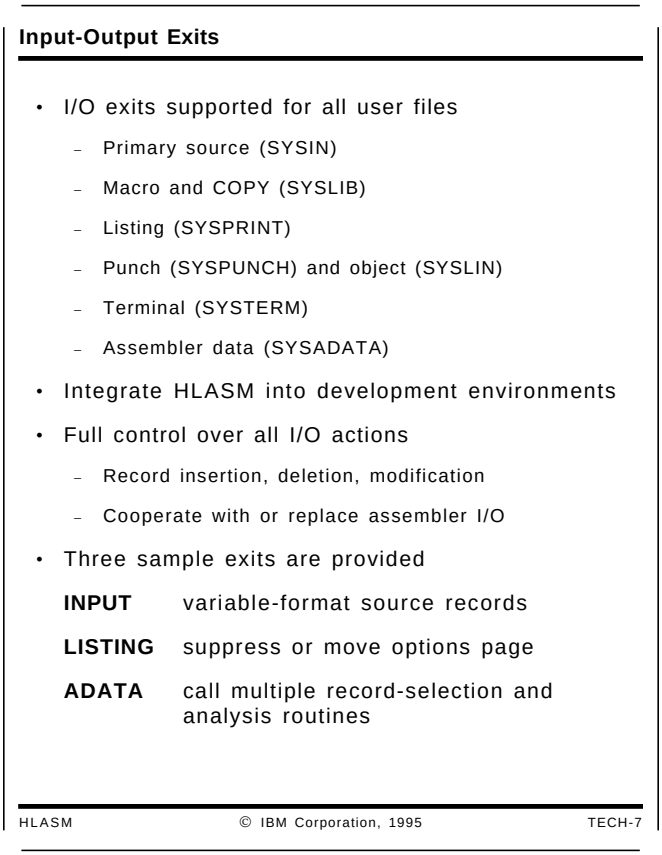
  ```
  &C      SETC   'ABCDE'(4,5)
  ```

  specifies a substring (five characters, starting with 'DE') that extends beyond the end of the original string; this is an error. Normally, HLASM issues the ASMA094W warning message; if FLAG(NOSUBSTR) is specified, HLASM suppresses the warning.

2.5.2 USING Warnings

The WARN(nn) suboption of the USING option enables checking for several (often obscure) but common USING-statement errors and oversights. Because USING statements are the most important (and probably, the most confusing) of the Assembler Language's addressing facilities, these additional features help with specifying them and diagnosing possible misuses:

- When a USING statement overrides (or nullifies) the effect of another, the associated warning message indicates the statement number of the other (or remote) USING statement involved in the override.
- A similar warning is provided when two USING ranges overlap, possibly indicating that an incorrect base register might be used. (An overlap of exactly one byte will not be flagged.)
- Warnings can be issued if the range of addressability of a base register exceeds a threshold specified in the LIMIT suboption.
- The use of register zero as a base register, with a nonzero absolute base address, may be detected.
- New diagnostics are provided to help locate errors involving qualified USING statements.



2.6 Input-Output Exits

Exits are supported for all user input and output files:

- Source (SYSIN)
- Macro and COPY (SYSLIB)
- Listing (SYSPRINT)
- Punch (SYSPUNCH) and object (SYSLIN)
- Terminal (SYSTERM)
- Assembler data (SYSADATA).

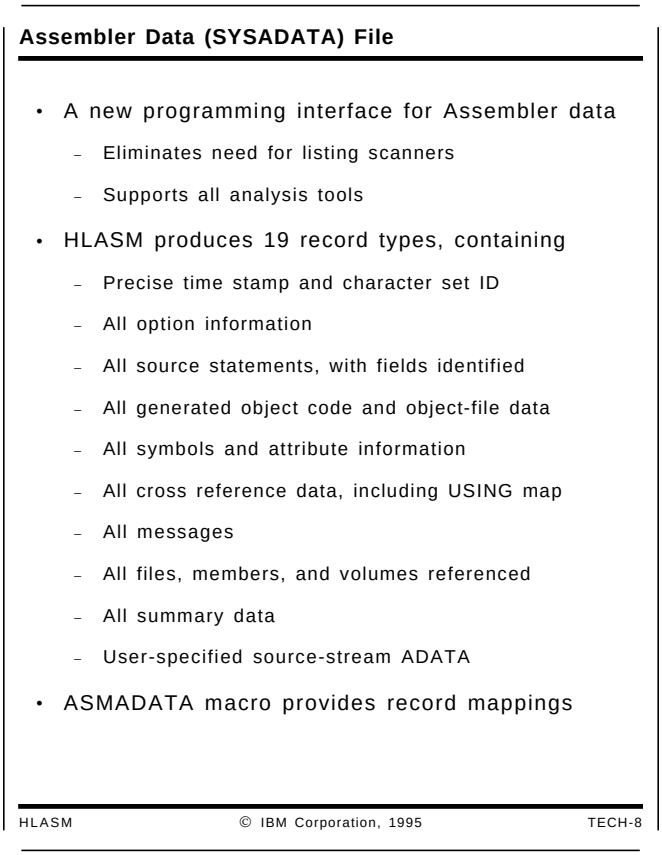
Each exit can:

- Delete, insert, or modify records as they flow to or from the assembler
- Supplement or supplant the Assembler's input and output facilities
- Insert informational messages of any severity into the Assembler's listing.

These exits provide complete control over the Assembler's input and output interfaces to its external files. They therefore greatly enhance the flexibility of integrating HLASM into programming environments with other tools such as librarians, code managers, program analyzers, configuration managers, listing reformatters, and debuggers.

A detailed description of the interface between High Level Assembler and the exits is provided in all editions of the HLASM *Programmer's Guide*. Mapping macros for the exit parameter lists and the following three sample exits are provided as optional installation materials:

- An INPUT exit (ASMAXINV) accepts variable-length source records, converting them to fixed-length 80-character records (with continuation records, if necessary).
- A PRINT exit (ASMAXPRT) supports options to suppress or move the options page and/or to suppress the summary page at the end of the assembly.
- An ADATA exit (ASMAXADT) supports one or more filter routines that may inspect or ignore selected ADATA records as HLASM produces them, and extract information for other uses.



2.7 Assembler Data (SYSADATA) File

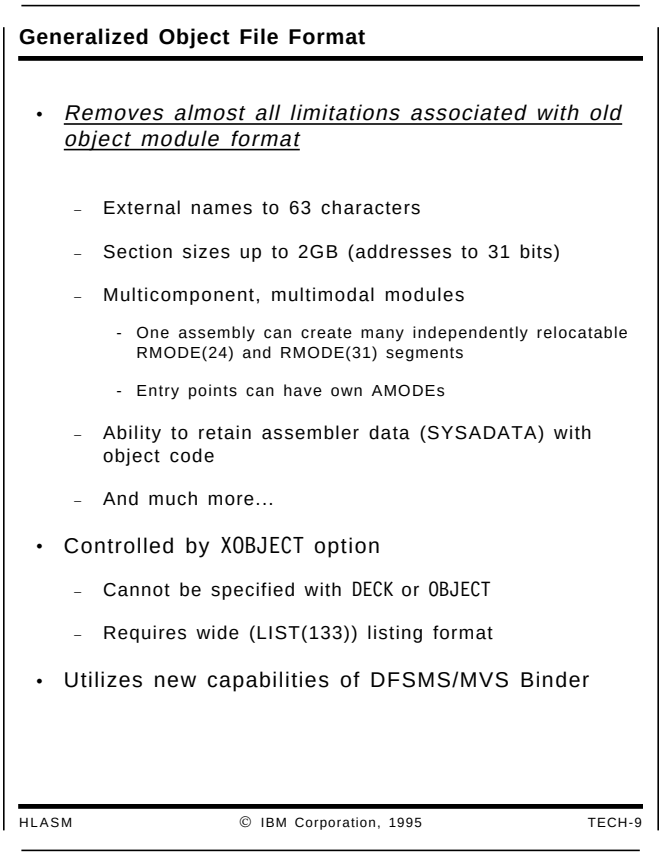
The ADATA option causes HLASM to produce records containing information about all aspects of the assembly. These ADATA records are intended as a stable programming interface for Assembler data. They eliminate any need for listing “scanners”; the availability of two listing formats in HLASM dictates the use of a reliable interface: the SYSADATA file.

Each assembly produces a sequence of records, started and ended with a pair of Compilation Unit records delimiting the boundaries of each assembly in a batch. Among the records are:

- An Identification record containing a precise time stamp and the coded character set ID (CCSID) used by the Assembler
- Option information describing all options used for the assembly
- Source statements, with the name, operation, operand, and remarks fields identified
- The generated object code, and object-file data such as the external symbol and relocation dictionaries
- Full symbol definition and attribute information
- All symbol and other cross reference data, including the USING map, the macro and COPY-member usage data, and the DSECT cross reference

- All diagnostic messages issued by HLASM (or exits), with information to help locate the flagged statement
- All input and output files, identified by data set name, member name, and volume ID
- All assembly summary data
- Any user-specified source-stream ADATA, if ADATA instruction statements are present.

This Assembler-data file increases the usability of HLASM by making it simpler to support programming tools and program development environments.



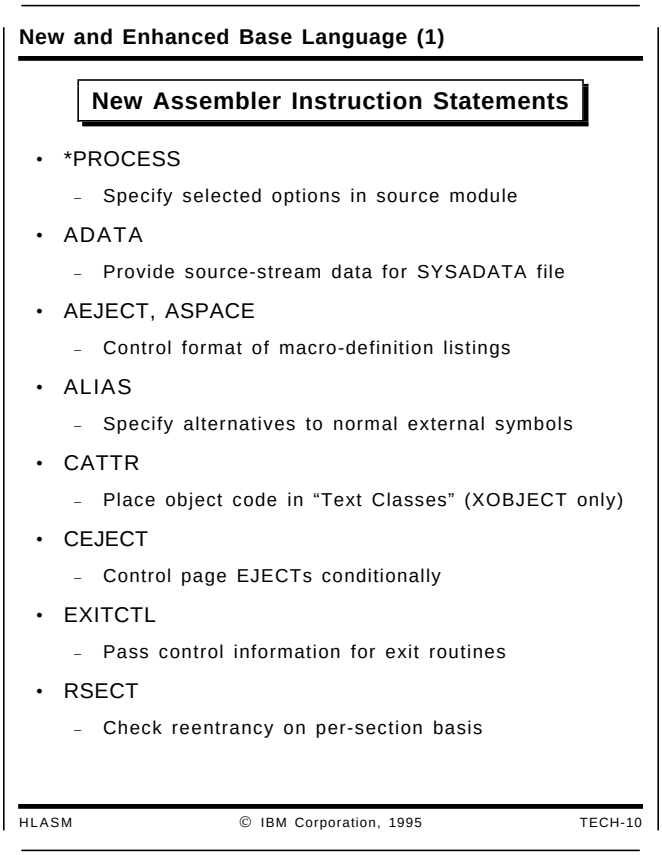
2.8 Generalized Object File Format

If requested by the XOBJECT option, High Level Assembler creates a new Generalized Object File Format. This extended object file supports such enhancements as external names longer than eight characters and control sections and other external objects larger than 16MB. It is processed by a forthcoming release of the DFSMS/MVS Binder.

Among the enhancements that the XOBJECT option provides are:

- External names up to 63 characters long
- Section lengths up to 2GB, and addresses and lengths 31 bits long
- Multicomponent, multimodal modules, with a single assembly capable of producing independently relocatable and independently loadable segments with different RMODEs
- AMODE attributes may be assigned to ENTRY points (not just to control section names)
- Assembler data (SYSADATA) may be included in the object stream, allowing both the object code and all associated descriptive data to be kept together in one program object.

XOBJECT requires a wide listing format, specified implicitly or explicitly, by either the LIST(133) option or LIST(MAX) with a print-line record length of at least 133 characters. The new and old object module formats are mutually exclusive: XOBJECT cannot be specified with either DECK or OBJECT.



2.9 New and Enhanced Base Language (1)

HLASM provides a special extension to the comment statement, and eight new Assembler instruction statements.

***PROCESS**

The *PROCESS statement allows you to assign module-specific options in the source statements of each assembled module. Further details are discussed at 2.3.4, “Source-File Options” on page 25.

ADATA

The ADATA statement allows you to create source-stream data to be inserted by HLASM into the SYSADATA output stream. This information could be produced by a program editor, a preprocessor, or the programmer.

ASPACE, AEJECT

The ASPACE and AEJECT statements permit spacing and page ejects among the lines of a macro definition, which helps to improve its readability. (They are not model statements; when the macro is called, the ASPACE and AEJECT statements do not appear in the generated code.)

ALIAS

The ALIAS statement for external symbols changes a syntactically valid external symbol to another character string in the object module. This permits Assembler output modules to be linked with those from other

languages whose external symbols contain characters that would otherwise be syntactically invalid in the “normal” Assembler Language.

The ALIAS statement permits 64-character external names when the XOBJECT option is active.

CATTR

The CATTR statement, when used in combination with the XOBJECT option, controls the placement of machine language instructions and data into multicomponent executable modules called *program objects*.

CEJECT

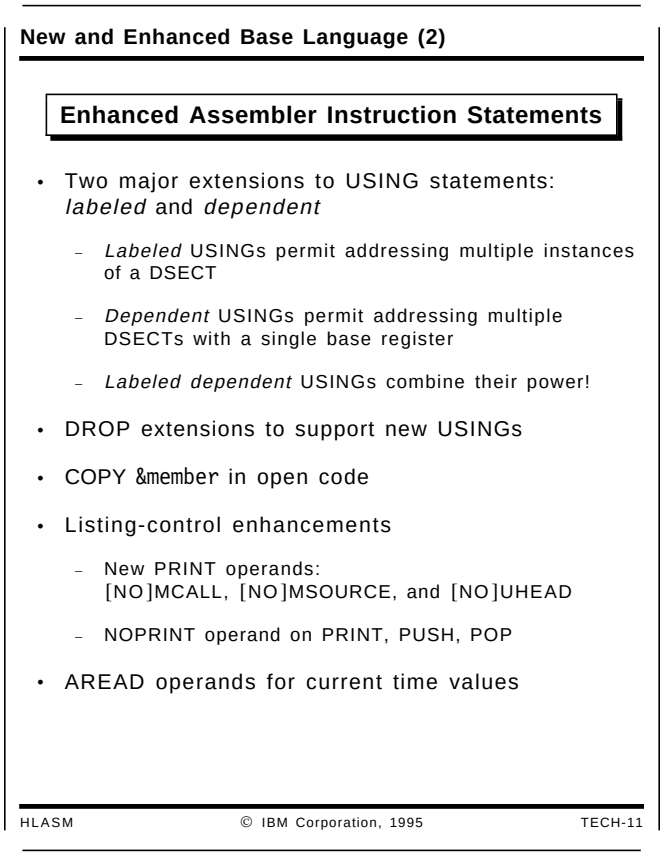
A conditional page-EJECT statement, CEJECT, permits automatic determination of the amount of space remaining on a listing page, with a page skip occurring if less than a requested amount remains. This removes the necessity of manually counting lines to determine where to put EJECT statements.

EXITCTL

The EXITCTL statement passes data to exit routines to enhance the flexibility of managing the behavior of I/O exits.

RSECT

The RSECT statement permits reentrancy checking on a per-section basis; thus an assembly may contain a mixture of reentrant and nonreentrant control sections. (RSECT was undocumented and inconsistently implemented in Assembler H; the HLASM implementation has been clarified and corrected.)



2.10 New and Enhanced Base Language (2)

HLASM adds many important new capabilities to existing statements:

- There are two major extensions to the USING statement: *labeled* and *dependent*. They may be used in combination, as *labeled dependent* USINGs.

  This powerful enhancement permits much greater control over the assignment and resolution of base addresses in symbolic expressions and provides a capability that can substantially improve the reliability of Assembler Language applications.

  – Labeled USINGs permit you to address multiple instances of a DSECT without the usual additional USING and DROP statements, and without the need to explicitly code offsets and base registers. Thus, you can concurrently manage multiple copies of the same DSECT-defined data structure using the full symbolic capabilities of the Assembler Language.

  – Dependent USINGs permit you to address multiple DSECTs that are anchored by a single base register, enabling you to describe nested code or data structures. Thus (unlike the symbolic addressing techniques required with all previous assemblers) you can actually reduce the number of general registers required for addressing DSECTs and assign them to other uses. This permits you to write more efficient

code while retaining the traditional advantages of fully symbolic addressing for DSECT-mapped data.

  - Labeled dependent USINGs combine the benefits of both extensions. You could, for example, describe record structures containing multiple instances of nested substructures, or of substructures that depend on a variable elsewhere in the containing structure.

  These USING enhancements allow you to easily manage complex data structures that are commonly used in higher level languages such as COBOL, PL/I, Pascal, or FORTRAN. Previous assemblers could describe those structures only with very complex and difficult coding.

- The DROP statement has been extended in support of the above enhancements to the USING statement.
- The COPY statement has been enhanced to permit specifying a variable symbol as its operand when the statement appears in "open code." Thus a program can selectively COPY program segments depending on other variables and controls and can avoid errors due to attempts to COPY nonexistent members.
- There are two enhancements to listing controls:
  - The PRINT statement supports the new [NO]MCALL, [NO]MSOURCE, and [NO]UHEAD operands. These provide dynamic, localized control over the contents of the output listing.
    - The MCALL operand causes the Assembler to display the actual text of the calling statements for inner macro calls. This permits you to easily debug complex nested macro interactions. (Macro calls displayed by the MCALL facility are not affected by the COMPAT(MACROCASE) option.)
    - The NOMSOURCE operand suppresses the display of subsequent macro-generated source statements while still showing the generated object code.
    - The UHEAD operand controls the presence of the USING heading at the top of each page.
  - The NOPRINT operand of the PRINT, PUSH, and POP statements can help to eliminate distracting detail in the listing due to uninteresting generated statements and makes it easier to use HLASM as a "cross-assembler" for other hardware architectures.
- The AREAD statement operands, CLOCKB and CLOCKD, return binary and decimal time and date information, respectively. These values return the *current* time (unlike the &SYSTIME variable symbol, which contains the time at which the assembly started and does not vary during the assembly).

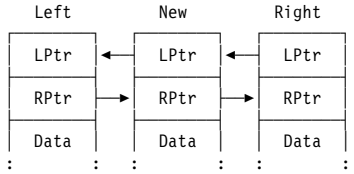
Note to the Presenter

The next four foils provide technical details of the new USING statements and diagnostics. If questions about these new features are important to your audience, the foils will help to illustrate how they may be used. If your audience does not want this greater level of detail, you may omit them and continue with 2.12, "New and Enhanced Base Language (3)" on page 46.

Labeled USINGs

- *Labeled* USINGs permit addressing multiple instances of a DSECT

  Example: Insert a New instance of Block in a doubly linked list between Left and Right elements




```
Block  DSect
LPtr   DS     A             Left sibling pointer
RPtr   DS     A             Right sibling pointer
Data   DS     XL273         Data area
```

  Three instances of the DSECT named Block are concurrently active:

```
RNew   Equ    5             R5 points to NEW

Left   Using  Block,2       Labeled Using
Right  Using  Block,3       Labeled Using
New    Using  Block,RNew    Labeled Using

       MVC    New.LPtr,Right.LPtr  Link New to Right
       ST     RNew,Right.LPtr      Link Right to New
       MVC    New.RPtr,Left.RPtr   Link New to Left
       ST     RNew,Left.RPtr       Link Left to New
```


Dependent USINGs

- *Dependent* USINGs permit addressing multiple DSECTs with a single base register

  Example: Three distinct control blocks reside in adjacent areas of storage, anchored with a single register.

```
CB1    DSect ,         Define control block 1
CB1F1  DS    D
CB1F2  DS    CL40
LCB1   Equ   * CB1     Length of block 1

CB2    DSect ,         Define control block 2
CB2F1  DS    24F
LCB2   Equ   * CB2     Length of block 2

CB3    DSect ,         Define control block 3
CB3F1  DS    XL1000

*      Get storage for all 3 Blocks, address in R7

       Using CB1,7           Anchor full storage block

*                            Next 2 USINGs are Dependent
       Using CB2,CB1+LCB1    Adjoin CB2 to CB1
       Using CB3,CB2+LCB2    Adjoin CB3 to CB2

       STM   14,12,CB2F1+12  Addresses resolved with
       XC    CB3F1(4),CB3F1  ...a single base register
```


Labeled Dependent USINGs

- *Labeled dependent* USINGs combine their benefits

  Example: code and two DCB mappings addressed with a *single* base register

```
       Using *,12

InDCB  DCB   DDNAME=..., etc.
OutDCB DCB   DDNAME=..., etc.

In     Using IHADCB,InDCB   Labeled dependent Using
Out    Using IHADCB,OutDCB  Labeled dependent Using

*      Following addresses are all resolved via R12
       MVC   Out.DCBLRECL,In.DCBLRECL

       DCBD  DSORG=PS       Generate IHADCB DSect
```

- Each instance of IHADCB DSect is “anchored” on a DCB
- Both DSects are active simultaneously
- All code and DSect addressing based on R12

USING Diagnostics

- USING in statement 3 nullifies USINGs in 2 and 4

```
                          1 START  CSect
       R:A  00000         2        Using *,10
       R:B  00000         3        Using *,11
ASMA301W ** WARNING ** Prior active USING on statement
                       number 2 overridden by this USING
       R:9  00000         4        Using *,9
ASMA300W ** WARNING ** USING overridden by a prior active
                       USING on statement number 2
```

- Message for statement 6 requested by specifying USING(LIMIT(X'F00')) option

```
 4120 BFFA         00FFA     6        LA    2,START+4090
ASMA304W ** WARNING ** Displacement exceeds limit value
                       specified
```

- Overlapping USING ranges

```
       R:7  00004         8        Using *,7
ASMA303W ** WARNING ** Multiple address resolutions may
                       result from this USING and the USING
                       on statement number 4
```

- R0 is only sometimes a valid base register!

```
                   00002    10 B      Equ   2
                   00000    11        Using B,0
ASMA302W ** WARNING ** USING specified Register 0 with a
                       nonzero absolute or relocatable
                       base address
```

2.11 Examples of New USING Statements and Diagnostics

The four examples on these foils illustrate some of the power of the new USING-statement facilities and USING diagnostics.

2.11.1 Labeled USINGs

In this example, three distinct instances of a control block named BLOCK are addressable at the same time, using registers 2, 3, and 5. (Previous assemblers could address only one instance at a time.) References to the left- and right-pointer fields, LPTR and RPTR, are *qualified* through the use of the qualifying symbols LEFT, RIGHT, and NEW. In this small code sequence, the NEW element (pointed to by register 5) is inserted into the list between elements LEFT and RIGHT. (We assume that their addresses have been previously placed into registers 2 and 3, respectively.)

Without labeled USINGs, the code for these operations would be much more convoluted and difficult to read, understand, and maintain.

2.11.2 Dependent USINGs

In this example, we assume that a large block of working storage will be acquired, and that it will contain several different independently defined data structures or control blocks named CB1, CB2, and CB3 in contiguous segments of the acquired storage. The dependent USING statements allow all of the control blocks to be addressed with a single register.

Previous assemblers required using a separate register to address each control block. Because several such blocks can now be referenced through a single register, registers previously required for addressing can be allocated to other useful purposes, thereby increasing the efficiency of the program.

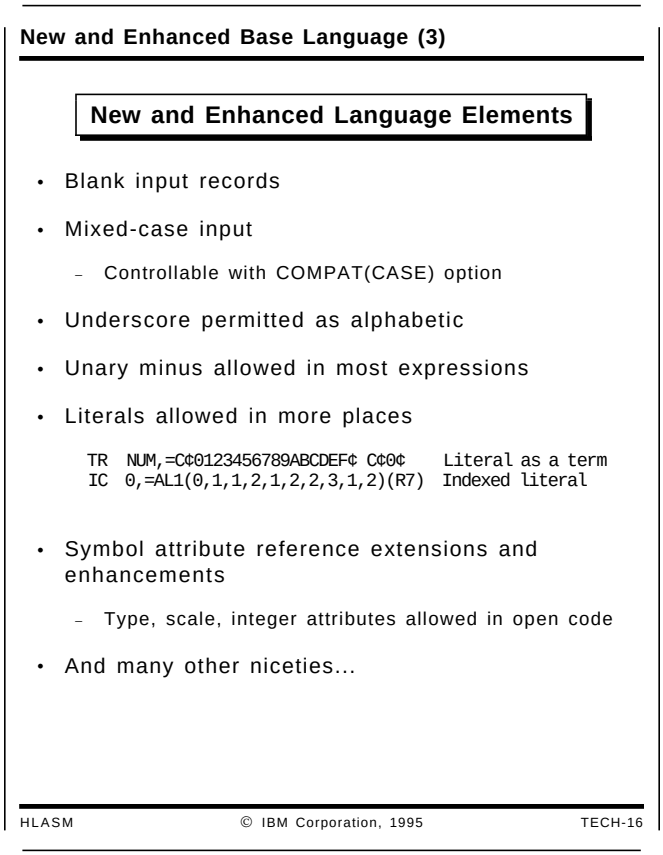
2.11.3 Labeled Dependent USINGs

This small example shows how one might combine the benefits of labeled and dependent USINGs in a single program. It assumes that there are two data control blocks (DCBs) addressable in the same program as the code and other items, and that we want to make symbolic references to fields in both DCBs at the same time. The labeled dependent USINGs permit fully symbolic references to both DCBs at the same time, and without needing additional registers.

As you can see from these three USING examples, the possibilities for mapping and addressing complex data structures are much richer and more varied than with previous assemblers.

2.11.4 USING Diagnostics

This small fragment of a program illustrates how the High Level Assembler detects possible error conditions. The first two messages warn of a common (and frequently erroneous) situation that previous assemblers did not detect. The third message can be requested when you suspect that some portions of your program are approaching the limits of addressability. The fourth warns of possible overlapping USING ranges, and the last warns of an attempt to use General Register zero as a base containing a nonzero address.



2.12 New and Enhanced Base Language (3)

HLASM has many new language features. Some of the more important and useful are:

- Blank input records are permitted and can also be encoded into macro definitions as model statements. (This frequently requested capability considerably improves the readability of programs.)
- Mixed-case symbols and operation codes are accepted: case is ignored internally for those items, so that all forms of a given symbol are equivalent. (If the uppercase-only behavior of previous assemblers is required, you may specify the COMPAT(CASE) option.)
- The underscore character is permitted as an alphabetic character.
- The unary minus operator is allowed in most arithmetic expressions.
- Literals may be used as relocatable terms in expressions (but not in expressions appearing in DC or DS statements). As symbols, they may be indexed in RX-type instruction statements. These extensions relax restrictions in previous assemblers and simplify situations where literals provide greater flexibility and ease of use. For example, in the instructions:

```
TR  NUM,=C'0123456789ABCDEF'-C'0'    Literal as a term
IC  0,=AL1(0,1,1,2,1,2,2,3,1,2)(R7)  Indexed literal
```

the first literal is used as a relocatable term in an expression; the second literal is used in an RX-type instruction as an address indexed by register 7.

Note to the Presenter

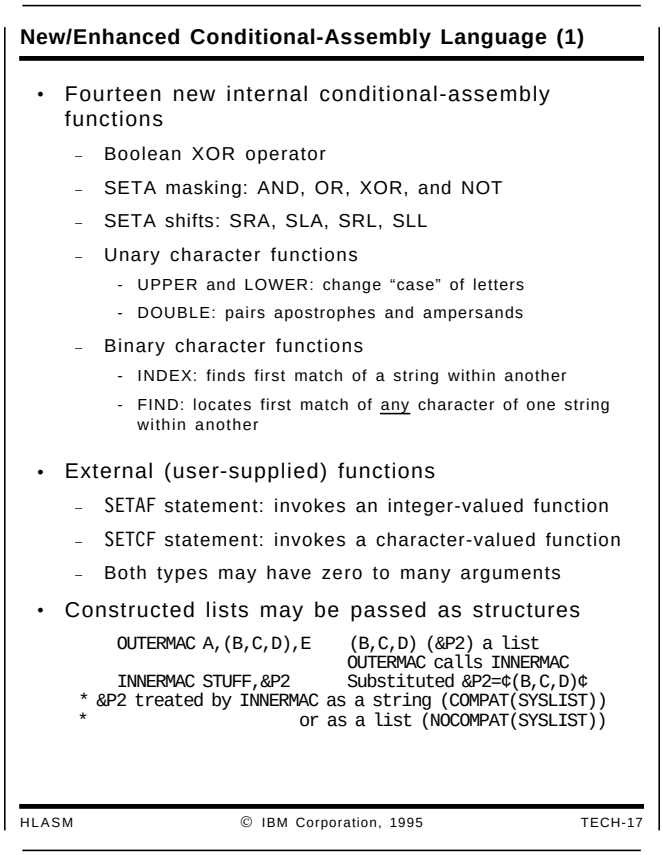
In case you are asked what these instructions do:

- The use of the first literal is typical of instructions that unpack a string of bytes into hexadecimal digits (one per byte) and then translate the bytes to characters for display.
- The second instruction replaces the rightmost byte in Register 0 with a count of the number of one-bits in Register 7 (assuming the value in R7 lies between 0 and 9).

- Symbol attribute references for type (T′), length (L′), scale (S′), and integer (I′) are allowed for ordinary symbols, SETC symbols, and literals. All may appear in conditional-assembly and other statements, in both macros and open code. This permits greater flexibility in constructing statements with greater degrees of parameterization, and it generalizes a capability previously available only inside macros.

  Note: Some unusual operands using character strings starting with a single letter followed by an apostrophe might be recognized by HLASM as attribute references where previous assemblers had ignored them.

- Other helpful enhancements include the following:
  - In the CNOP statement, symbols in operands are not always required to be previously defined.
  - Symbol resolutions for EQU statements relax some previous requirements that certain symbols be previously defined.



2.13 New and Enhanced Conditional-Assembly Language (1)

HLASM supports two types of conditional-assembly functions: internal (built-in) and external (user-written). Both provide enormous simplification in writing common but complex data manipulations as well as the potential for making assemblies run faster.

2.13.1 Internal Conditional-Assembly Functions

Fourteen new internal functions let you write conditional-assembly statements, such as in macros that perform common programming functions, with greater ease and efficiency. These 14 new functions include:

- Boolean operation: XOR between Boolean expressions
- Masking: AND, OR, XOR, and NOT on fullword binary arithmetic data (SETA expressions)
- Shifting: left and right arithmetic and logical shifts (the SRA, SLA, SRL, and SLL functions are exactly equivalent to the similarly named machine instructions) of fullword binary data (SETA expressions).

- Unary character functions
  - The UPPER and LOWER functions change the case of the letters in a character string to uppercase and lowercase, respectively.
  - The DOUBLE function checks the string for apostrophes and ampersands and replaces each with a pair, allowing direct substitution into DC statement operands and literals.
- Binary character functions
  - The INDEX function finds the first match of one string within a second string.
  - The FIND function locates the first match of any character from one string within a second string.

2.13.2 External Conditional-Assembly Functions

Two new statements are provided to support calls to external functions:

SETAF

The SETAF conditional-assembly statement is used to invoke an external (user-written) function of arithmetic type, with any number of integer arguments. For example, the statement:

```
&IntVar  SETAF  'IntFunc',&I,23
```

calls the external function IntFunc with two integer arguments, &I and 23.

SETCF

The SETCF conditional-assembly statement is used to invoke an external (user-written) function of character type, with any number of character arguments. For example, the statement:

```
&CharVar SETCF  'CharFn&J','&C','A*23BX'
```

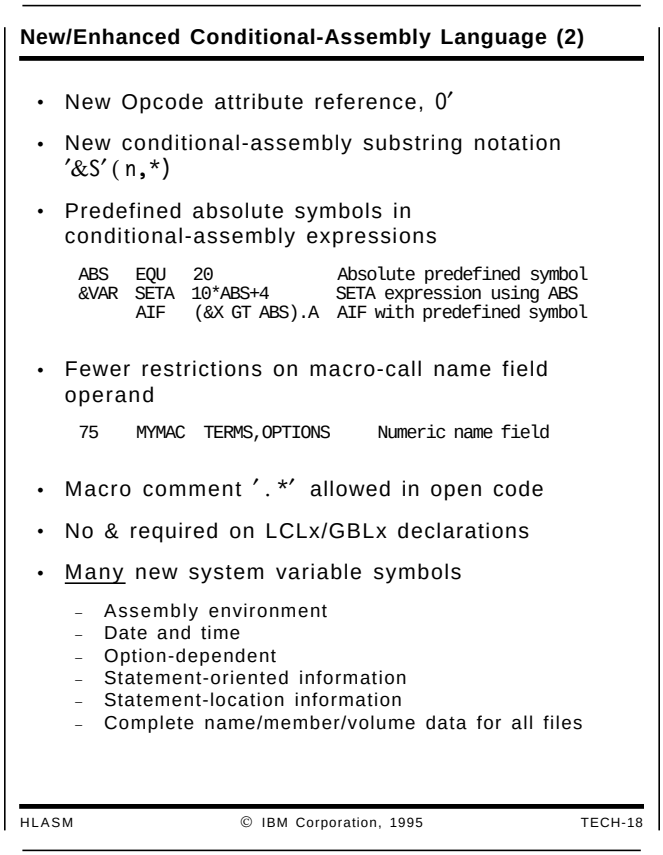
calls the external function whose name is constructed from the characters CharFn suffixed with the value of the variable &J, passing two character arguments, &C and A*23BX.

The external function capability allows you to write your own functions to operate on conditional-assembly data. External functions can perform any desired action, such as accessing operating system services, interacting with the programmer, reading or writing external files, or even replacing existing functions that were difficult or inefficient (or even impossible) to code in the older conditional-assembly language.

2.13.3 Inner Macro Arguments and the COMPAT(SYSLIST) Option

Previous assemblers always treated arguments passed from outer to inner macros as simple, unstructured character strings. Thus, inner macros had to parse the operands one character at a time. You may enforce this behavior in HLASM by specifying the COMPAT(SYSLIST) option.

However, if you specify the NOCOMPAT(SYSLIST) option, HLASM can recognize substituted sublists as having a list structure. Thus you can construct complex macro operands in an outer macro to be passed as list structures to inner macros. This capability can help remove many unnecessary distinctions between outer and inner macros.



2.14 New and Enhanced Conditional-Assembly Language (2)

These new and enhanced conditional-assembly language features assist developers and maintainers in many ways:

- A new 0′ attribute reference lets you check an operation-code mnemonic for possible usage. This capability can help you to determine which opcodes have been defined, or which library members may be accessible by macro calls or COPY operations. The possible values of this attribute are:

  A Assembler instruction (such as TITLE, SPACE)
  E Extended mnemonic (such as BNM, BZ)
  M A known, defined macro
  O Machine instruction (such as L, ST)
  S A member of this name exists in SYSLIB
  U Undefined or unknown.

- A new notation has been introduced for substrings in the conditional-assembly language. The length of the substring need not be explicitly specified, as in the previous notation:

  ```
  &C      SetC   '&CharVar'(&Start,K'&CharVar-&Start+1)
  ```

  Instead, the new * notation for the default substring length:

  ```
  &C      SetC   '&CharVar'(&Start,*)
  ```

  indicates “from here to the end of the string.”

- Extensions to permit use of predefined absolute symbols in conditional-assembly expressions (such as SETA, SETB, SETC, and AIF) encourage a richer interaction between the ordinary Assembler Language and the inner or conditional-assembly language. For example, you can now use symbolic values defined by EQU statements in much more general ways to control and parameterize conditional-assembly operations.
- The name-field entry of a macro call instruction may be treated more flexibly as an operand. This extension permits a more symmetric treatment of name-field and operand-field parameters to macro calls.
- Macro-comment statements (starting with the two characters '.*') are now allowed in open code and are ignored in look-ahead mode.
- In LCLx and GBLx statements declaring local and global variable symbols, the requirement for an ampersand to be prefixed to the symbol has been removed.
- HLASM provides a wide variety of new system variable symbols, whose values characterize many aspects of the internal and external assembly environment. These are described in detail in 2.15, "System Variable Symbols" on page 53.

Note to the Presenter

You can omit the following discussion of system variable symbols if your audience does not want this level of detail.

System Variable Symbols

1. Assembly-environment variables
   - &SYSASM: Name of the Assembler
   - &SYSVER: Assembler version, release, and modification level
   - &SYSTEM_ID: Operating system under which this assembly is being performed
   - &SYSJOB, &SYSSTEP: Assembly job and step name
   - &SYSPARM: Assembler invocation parameter(*)
2. Date and time variables
   - &SYSDATC: Assembly date (format YYYYMMDD)
   - &SYSDATE: Assembly date (format MM/DD/YY)(*)
   - &SYSTIME: Assembly start time (format HH.MM)(*)

(*) Existed in Assembler H

System Variable Symbols ...

3. Option-dependent variables
   - &SYSOPT_DBCS: DBCS option setting
   - &SYSOPT_RENT: RENT option setting
   - &SYSOPT_OPTABLE: Name of the operation-code table used for the assembly; set by the OPTABLE option
4. Statement-related variables
   - &SYSSEQF: Contents of the sequence field of the current input statement
   - &SYSSTMT: Number of the next statement to be processed by the Assembler
5. Current statement-location variables
   - &SYSLOC: Name of current location counter(*)
   - &SYSSTYP: Type of the current control section into which statements are being grouped
   - &SYSNEST: Nesting level at which the current macro was invoked (macros called from open code are at level 1)

(*) Existed in Assembler H

System Variable Symbols ...

6. File-information variables (three variables per file)
   - &SYS**xxx**_DSN: data set or file name
   - &SYS**xxx**_MEMBER: member name (if any)
   - &SYS**xxx**_VOLUME: volume identification

xxx	**Input Files**
IN	Current primary input file
LIB	Current library input file

xxx	**Output Files**
PRINT	Listing file
PUNCH	Object-module file
LIN	Object-module file
ADATA	SYSADATA file
TERM	Terminal-display file

2.15 System Variable Symbols

Thirty-four new system (&SYS) variable symbols are available. They provide conditional-assembly access to a rich variety of information about the assembly and its environment, allowing you both greater control over the generated object code, and the ability to embed useful information about the assembly into the object code for configuration management and diagnostic aids.

Note: HLASM supports all of the system variable symbols listed below. Assembler H supported four of them: &SYSDATE, &SYSTIME, &SYSSECT, and &SYSPARM.

- Assembly-environment variables

  &SYSASM, &SYSVER
  : These variables provide the name of the Assembler and its version, release, and modification level.

  &SYSTEM_ID
  : This variable provides an identification of the operating system under which the current assembly is being performed.

  SYSJOB, &SYSSTEP
  : These variables provide the job name and job-step name under which the Assembler is running.

  &SYSPARM
  : This variable provides the character string “xxx” supplied in the SYSPARM(xxx) option when the Assembler was invoked.

- Date and time variables

  &SYSDATC
  : This variable provides the current date in the form YYYYMMDD.

  &SYSDATE
  : This variable provides the current date, in the form MM/DD/YY.

  &SYSTIME
  : This variable provides the time at which the assembly started, in the form HH.MM.

- Option-dependent variables

  &SYSOPT_DBCS
  : This binary variable provides the setting of the DBCS option. Macros might need to generate different code in DBCS contexts.

  &SYSOPT_OPTABLE
  : This character variable provides the name of the current operation code table used for this assembly, as established by the OPTABLE option.

  &SYSOPT_RENT
  : This binary variable provides the setting of the RENT option. Macros might need to generate different code for reentrant and nonreentrant situations.

- Statement-related variables

  &SYSSEQF
  : This variable provides the contents of the sequence field of the current input statement. This information can be used for debugging data.

  &SYSSTMT
  : This variable provides the number of the next statement to be processed by the Assembler. Debugger data that depends on statement numbers can be generated with this variable.

- Current statement-location variables

  &SYSLOC
  : This variable provides the name of the current location counter (typically, the name of the current control section).

  &SYSNEST
  : This arithmetic variable provides the nesting level at which the current macro was invoked (the outermost macro is at level 1; open code is at level 0).

  &SYSSTYP
  : This variable provides the type of the current control section into which statements are grouped or assembled (CSECT, DSECT, or RSECT) when a macro is invoked. If a macro must generate code in a different control section, this variable permits the macro to restore the previous environment before exiting.

- Assembler input-output file variables

  Each of the Assembler′s input and output files has three associated variables: the name of the file or data set, the member name (if any), and the volume identifier for the volume on which the data set resides. There are seven user input-output files; not all of them may be opened, depending on the assembly options in effect:

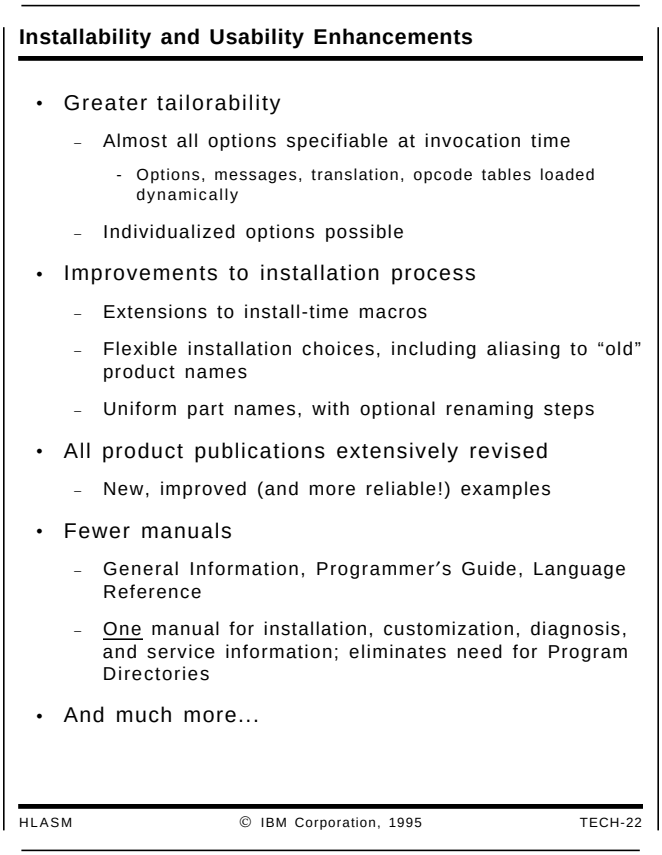
IN	Current primary input file (SYSIN)
LIB	Current library input file (SYSLIB)
PRINT	Listing file (SYSPRINT)
PUNCH	Object module file (SYSPUNCH)
LIN	Object module file (SYSLIN)
ADATA	Assembler data file (SYSADATA)
TERM	Terminal file (SYSTERM)

  The characters **xxx** below are replaced by the two to five identifying characters from the list above.

  &SYS**xxx**_DSN
  : The name of the current file or data set.

  &SYS**xxx**_MEMBER
  : The member name (if any) of the current file or data set.

  &SYS**xxx**_VOLUME
  : The name of the current volume for this file or data set.



2.16 Installability and Usability Enhancements

Usability of HLASM has been enhanced in several additional ways:

- HLASM makes most Assembler features selectable by options, rather than being fixed at product installation time.

  Key elements of the assembly environment—the default options table, the messages table, the translation table, and the operation-code table—are loaded dynamically at assembly invocation. Thus it is less necessary to choose a single set of global default options for all users at installation time to satisfy the needs of many different users.

- Because HLASM dynamically loads the default options module at the start of the assembly, users can create their own individualized options modules. Resources are saved because it is unnecessary to generate separate Assemblers with different defaults or specify option overrides on each assembly.

The installation process has been made more flexible:

- The install macros permit the HLASM installer to specify a greater variety of features, and that certain options may not be overridden when invoking the Assembler. This helps to enforce organizational coding standards and prevent errors due to incorrectly chosen options.

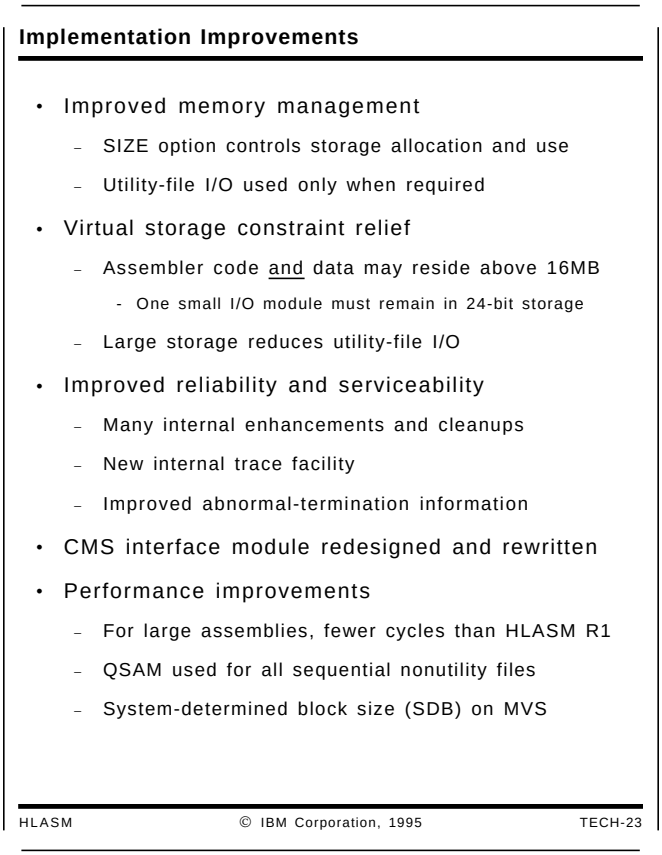
- An optional installation step allows you to assign to HLASM Release 2 appropriate “old-product” aliases for ASMA90 and HLASM. The alias for ASMA90 is IEV90 (for Assembler H under MVS and CMS), and the alias for HLASM is HASM (for Assembler H under CMS).
- Part names have been regularized: all start with the characters ASMA, and an optional install step is provided to allow you to rename the items to their (occasionally nonstandard) HLASM Release 1 values.

The Assembler publications have been improved:

- All product publications have been extensively revised and upgraded. There are new and additional examples as well as some sample programs to assist in the use of the new capabilities.
- The *IBM High Level Assembler/MVS & VM & VSE, Release 1 Installation* manual has been completely revised in HLASM R2 to contain all installation, customization, diagnosis, and service information. Thus, the number of manuals has been reduced, and there is no longer any need for Program Directories.

Among other improvements are:

- HLASM can be installed in shared storage on MVS, CMS, and VSE/ESA. For many small, or for batched, assemblies, this eliminates the need for the multiple phases of the DOS/VSE Assembler, or for the “managed overlays” used in Assembler H. Thus, small- and medium-sized assemblies may run more efficiently.
- The Installation Verification Sample program has been updated to use some of the new statements in HLASM.
- Mapping macros are provided for SYSADATA records, external functions, and I/O exit work areas.
- Installation support is provided for VM systems only with VMSES/E.
- Under CMS, logical segment (LSEG) support has been added. When installing HLASM, you may specify that searches for the Assembler should (a) search segments only, (b) search disk only, or (c) search segments first followed by a disk search if the Assembler cannot be found in shared segments.



2.17 Implementation Improvements

Many features of HLASM′s implementation are not directly visible but assist developers and maintainers in many ways:

- SIZE option

  Previous assemblers used all available storage; the new SIZE option permits much greater control over the amount of storage acquired by the Assembler. Thus users can invoke the Assembler under the control of other monitors and job-flow programs, enabling the Assembler to share storage with concurrent processors if desired.

- Improved memory management and reduced utility-file I/O

  Previous assemblers use utility files either by necessity of design or when they do not need to; HLASM uses central storage whenever possible and does work-file I/O only when necessary.

- Virtual storage constraint relief

  HLASM Release 2 can utilize storage above 16MB, thus reducing pressures on scarce 24-bit-addressable storage and possibly eliminating the need for utility-file I/O.

  The Assembler itself can be placed in 31-bit storage; only a small I/O interface module must remain below 16MB.

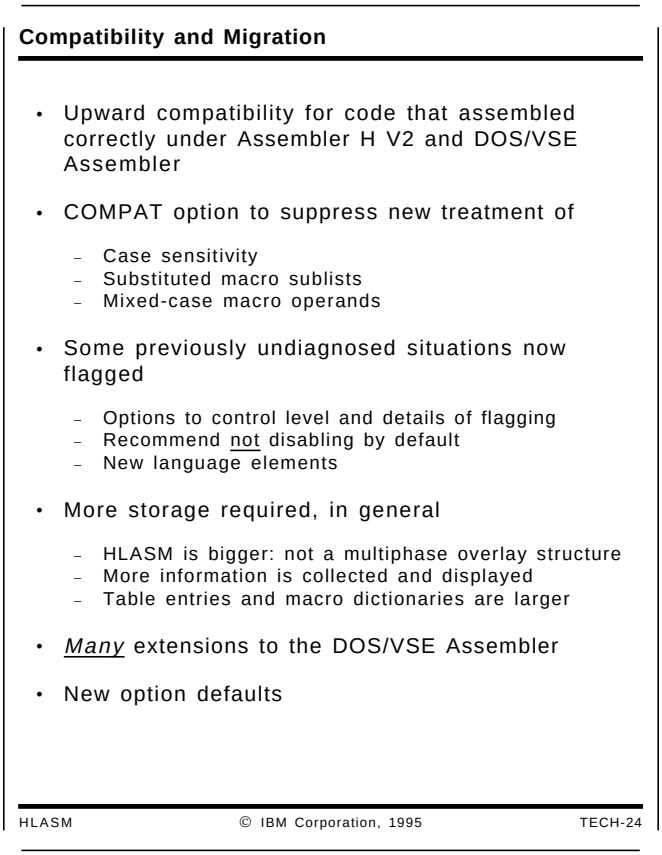
- High Level Assembler is based on Assembler H Version 2.1 and contains major internal changes to data and control structures to improve its reliability, availability, and serviceability.
- Internal trace facility

  Previous assemblers had no capability (other than formatting certain work areas on abnormal termination) to assist maintainers in locating and correcting errors in the Assembler. HLASM Release 2 supports a new internal trace facility that service personnel can use to extract information about the internal behavior of the Assembler to detect and isolate internal problems efficiently and repair them expeditiously.
- Terminal error conditions that otherwise cannot be delivered to the user are written to the job log by the Assembler. Wherever they appear, these messages give more precise information than messages from previous Assemblers about the cause of internal problems or environmental conditions forcing abnormal termination.
- The CMS interface module has been revised and enhanced to support newer levels of the CMS operating system and to handle options better; some old, unsupported, and obsolete options have been removed.
- Performance improvements

  Assembler performance has been improved in many areas. In addition to the resource savings available from utilizing large amounts of central storage, processor utilization for large macro-based assemblies has been reduced compared to HLASM Release 1.

  - For assemblies with few or no macros, HLASM provides performance benefits in all areas.
  - For assemblies with heavy macro usage, CMS and MVS show that additional processor and elapsed time are required. The reason is that HLASM provides a much richer set of services and information for macros (for example, the 34 new system variable symbols); more processor time is needed to support them. Because machine resources are becoming less expensive at the same time that human resources with Assembler Language skills are becoming more expensive, it is a design factor for HLASM to “trade machine time for people time.” Thus, we expect that the added costs of machine time for assembling certain macro-based applications will be more than repaid by the savings in the time and effort needed to support them.

    Substantial improvements are provided in the VSE/ESA environment, in all cases.
- HLASM uses QSAM I/O for all sequential nonutility files, to obtain greater efficiency and increased reliability, availability, and serviceability. This capability makes improvements in access methods automatically available to the Assembler without recoding.
- On MVS systems, SDB is supported if the appropriate level of DFSMS/MVS is present.
- HLASM Release 1 uses a very simple test for RENT checking (namely, whether a literal may be used as an operand), which may occasionally cause inaccurate diagnostics. HLASM Release 2 uses separate flags for RENT and literal usage checking.



2.18 Compatibility and Migration

A key design consideration for HLASM is that existing code should continue to assemble correctly, with the appropriate options settings.

- Because the enormous existing base of Assembler Language code must continue to be supported, HLASM provides as full upward compatibility with previous assemblers as possible.

> **Note to the Presenter**
>
> You may want to skip the details on this and the next foil. However, if compatibility questions are important to your audience, the key factors are listed here; more details are covered in the next foil.

The possible compatibility differences are:

- The COMPAT option controls the High Level Assembler′s treatment of mixed-case unquoted macro arguments, macro sublists, and mixed-case input text. If the COMPAT option is set appropriately, the High Level Assembler will behave just as Assembler H (and, for cases not already documented as significant differences, the DOS/VSE Assembler) behaved.
- If you enable some of the new diagnostics, some conditions might be flagged that the DOS/VSE Assembler or Assembler H did not detect. Among these

are possible continuation errors and USING-statement overlaps and misuses. You can control this flagging with appropriate option settings.

- New language elements in HLASM may be recognized in contexts where previous assemblers ignored them. For example, the availability of new attribute references in open code may cause operand strings beginning with any of the pairs of characters T′, I′, or S′ to be flagged.
- In general, HLASM requires more storage (compared with previous assemblers) to handle the same jobs, for several reasons:
  - To allow the entire HLASM to execute in shared storage, it was organized as a single module, rather than as a multiphase self-managed overlay structure. The entire Assembler is reentrant and can be installed in shared storage, allowing assemblies in a smaller working storage.
  - More information is retained in internal tables, to allow HLASM to produce the new cross references and diagnostics.
  - The macro dictionaries hold more information (for example, the new system variable symbol values).
- The many enhancements over the DOS/VSE Assembler are too numerous to mention here; a full list is provided in each edition of the HLASM *Programmer's Guide*.

  OPTABLE(DOS)
  The OPTABLE option has been extended to accept a DOS suboption, which specifies that HLASM Release 2 should use an operation-code table containing instructions compatible with the DOS/VSE Assembler. This option is useful for migrating Assembler Language applications from DOS/VSE to VSE/ESA.

  The VSE/ESA 2.1 system automatically generates substitute job control statements in place of the // EXEC ASSEMBLY statement to invoke HLASM with options making it more nearly compatible with the VSE/AF and DOS/VSE Assemblers.
- New option defaults are provided that will increase ease of use and the reliability of assembled programs (see Table 1).

Table 1. Default Option Settings in HLASM Release 2, HLASM Release 1, and ASMH

HLASM Release 2	HLASM Release 1	Assembler H V2.1
BATCH	NOBATCH	NOBATCH
NODECK	DECK	DECK
FLAG(0,ALIGN,RECORD,CONT)	FLAG(0)	FLAG(0)
LANGUAGE(EN)	LANGUAGE(UE)	(Not available)
LINECOUNT(60)	LINECOUNT(60)	LINECOUNT(55)
LIST(121)	LIST	LIST
MXREF(SOURCE)	MXREF	(Not available)
OBJECT	NOOBJECT	NOOBJECT
NOPROFILE	(Not available)	(Not available)
NOTRANS	(Not available)	(Not available)
USING(WARN(15),MAP)	NOUSING	(Not available)
NOXOBJECT	(Not available)	(Not available)
XREF(SHORT,UNREFS)	XREF(FULL)	XREF(FULL)

Remember that HLASM does not support the old options described at 2.3.5, “Old, Unsupported Options” on page 26. Possible approaches to handling migration concerns caused by the changes in default options in HLASM Release 2 include the following:

- Change the invocation options

  Specifying FLAG(NOSUBSTR), FLAG(NOCONT), or USING(WARN(11)) will sometimes suppress unexpected diagnostics.

- Adding *PROCESS statements to modify assembly options

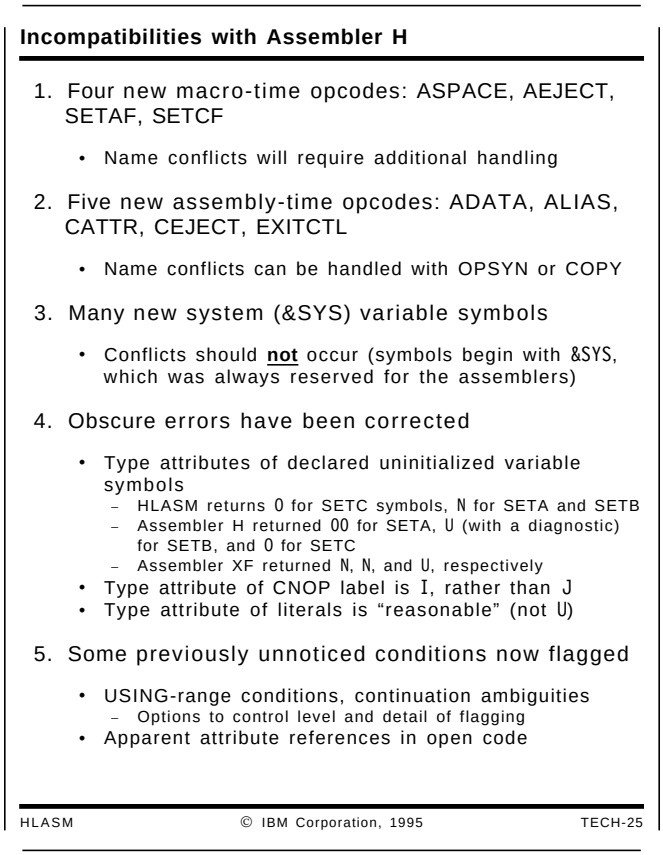
  Inserting the added statements to each affected module requires extra effort.

  Note: One of the main justifications for modifying the defaults in HLASM Release 2 to enable more diagnostics has been that customers have stated that the new diagnostics have great value.

- Review the affected parts

  Based on experiences with enabling the new diagnostics, we recommend that you consider carefully checking the code in which the messages appear.

- Note that install-time aliasing can help with migration from HLASM Release 1 to HLASM Release 2.



2.19 Incompatibilities with Assembler H

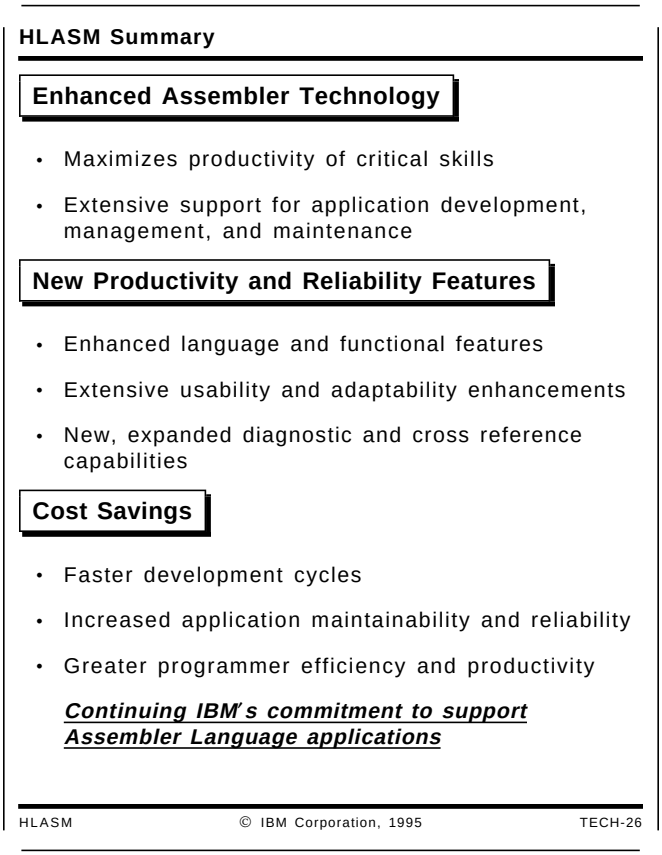
Some specific differences between HLASM and Assembler H are:

- If you used macros with the names of the nine new opcodes:

  ADATA, AEJECT, ALIAS, ASPACE, CATTR, CEJECT, EXITCTL, SETAF, SETCF

  it is possible that you might have to use OPSYNs or other bypasses to ensure that your macros are used instead of the new native opcodes.
- Many new system variable symbols have been introduced. If your program defines its own local or global variable symbols starting with the characters &SYS, it is possible that name collisions may occur. (But, remember that such &SYS variables have always been reserved for the assemblers!)
- Several inconsistencies in the evaluation of the attributes of undefined variable symbols have been corrected.
- A few minor (and very obscure) errors in the behavior of Assembler H have been corrected.

A comprehensive list of differences between HLASM and its predecessors is provided in Appendix A of each edition of the HLASM *Programmer's Guide*.



2.20 HLASM Summary

This presentation necessarily provides only a brief overview of the many new capabilities of HLASM.

The usability, language, listing, and other enhancements embodied in HLASM make development and maintenance of Assembler Language applications easier and efficient. Thus the savings in people time alone could justify the (small) effort required to make the transition from previous assemblers to HLASM.

In summary, HLASM provides greatly enhanced Assembler technology to:

- Support critical assembler-based applications
- Maximize the productivity of staff with critical Assembler skills
- Support integration with modern application development, debugging, and maintenance tools and environments.

You will find productivity gains in many areas:

- Numerous enhancements to the base language as well as to the inner conditional-assembly or macro language improve the readability, maintainability, and efficiency of Assembler Language applications
- Extensive usability enhancements improve the efficiency and productivity of Assembler Language programmers and maintainers

- Many new and expanded diagnostic capabilities facilitate delivering higher-quality and more reliable applications.

Using HLASM will result in cost savings attributable to:

- Faster development cycles due to performance and usability improvements
- More maintainable and reliable applications due to new language, diagnostic, and extensive support features
- More efficient and productive programmers.

IBM has provided a significant tool that can help Assembler application developers with every aspect of developing, debugging, and maintaining Assembler Language applications on its host System/360, System/370, and System/390 systems.

Chapter 3. Management Overview Presentation Foils

High Level Assembler:

Management Overview

Assembler Language Products
IBM Santa Teresa Laboratory
Software Solutions Division
555 Bailey Avenue
San Jose, California 95141

Topic Overview

IBM High Level Assembler for MVS & VM & VSE

- Why a new Assembler?
- Key benefits of HLASM
  - Organizational savings
  - Human resource savings
  - System resource savings
  - Tool and development environment support
- Compatibility and migration
- Summary

Why a New Assembler?

- Many customer applications contain critical components written in Assembler Language
  - Assembler components are still being developed, enhanced, maintained
  - Best vehicle for performance sensitivity, access to system functions, compact code (no run-time library!), application-specific language elements
- Assembler applications represent major investments
  - HLASM helps protect that investment
  - Can benefit from new features immediately
- Assembler programmers: a critical resource
  - Fewer people with Assembler Language skills; must maximize organizational benefits of their abilities and knowledge
  - HLASM targeted to needs of application developers and maintainers
- Continuing customer requirements for enhancements
  - Many are satisfied by HLASM
- Support for MVS/ESA, VM/ESA, and VSE/ESA with a single, modern, high-function Assembler

Organizational Savings

- Protect investments in applications
  - Improved support for existing code
  - Reduced costs of functional upgrades, corrective maintenance
- Protect investments in people and skills
  - Preserve existing application knowledge and project management experience
  - HLASM helps **all** aspects of development and maintenance
- Protect investments in procedures
  - Existing policies, procedures need not change
  - HLASM helps enforce organizational standards
  - Migration is quick and simple
- Avoid potentially significant costs of converting to new languages
  - Long learning curve for new language, compiler, library, and OS environment
  - Labor-intensive conversion efforts
  - Delays in deploying new function, plus new bugs

Human Resource Savings

- Many powerful new and expanded cross reference features

  → Less time and effort to manage applications of all sizes

- Extensive, numerous usability enhancements

  → Less time and effort on unproductive details

- Many new and improved diagnostics
  - HLASM helps detect common (but elusive and previously undetected) coding defects
  - More informative detail provided for many messages

  → Pinpoint errors earlier and more accurately

- Numerous language enhancements
  - Some can actually improve program efficiency!

  → Deliver applications faster, with higher quality and reliability

- Many new and/or enhanced options

  → Better process controls

System Resource Savings

- Performance improvements
  - Fewer processor cycles for almost all assemblies
- Better memory utilization, more flexible controls
  - Utilizes extended storage (“above the line”); SIZE option controls allocation
  - Reduced I/O′s and elapsed time
- Functional enhancements mean fewer reassemblies

Tool and Development Environment Support

- Optional “Assembler Data” file
  - Data about *every* aspect of the assembly
  - Basic data for program analyzers, debuggers, cross-referencers, and other tools
- Optional exits for all user files
  - Smoother integration with librarians, configuration managers, process controls, re-formatters
- Assembly-time external functions

Compatibility and Migration

- Upward compatibility
  - For code that assembled correctly under Assembler H V2 and DOS/VSE assembler: almost always 100%
  - Easy migration from old assemblers
- Some previously undiagnosed situations now flagged
  - Options control level and detail of flagging
- COMPAT options to suppress new treatment of
  - Case sensitivity
  - Substituted macro sublists
  - Unquoted mixed-case macro arguments
- Areas to check for possible differences
  - New opcodes: ADATA, AEJECT, ALIAS, ASPACE, CATTR, CEJECT, EXITCTL, SETAF, SETCF
  - Many new system (&SYS) variable symbols
  - Literals and attribute references
  - Obscure errors in previous Assemblers corrected

Enhanced Assembler Technology

- Preserves investments in code, people, processes
- Supports critical applications
- Maximizes productivity of critical skills
- Supports modern tools and environments
- Avoids conversion costs

New Productivity and Reliability Features

- Enhanced language and functional features
- Extensive usability and adaptability enhancements
- New, expanded diagnostic and cross reference capabilities

Cost Savings

- Faster development cycles
- Increased application maintainability and reliability
- Greater programmer efficiency and productivity

Continuing IBM's commitment to support Assembler Language applications

Chapter 4. Technical Overview Presentation Foils

High Level Assembler:

Technical Overview

Assembler Language Products
IBM Santa Teresa Laboratory
Software Solutions Division
555 Bailey Avenue
San Jose, California 95141

Topic Overview

IBM High Level Assembler for MVS & VM & VSE

- Key features
  - Assembler “externals”
  - Base language enhancements
  - Conditional-assembly language extensions
  - Installability enhancements
  - Implementation enhancements
- Compatibility and migration
  - Incompatibilities with Assembler H
- Summary

Key Features

- New and enhanced options
- Listing enhancements
- Diagnostics enhancements
- Input-output exits
- Assembler data (SYSADATA) file
- Generalized object file format
- New and enhanced base language
  - Examples of new USING statements and diagnostics
- New and enhanced conditional-assembly language
  - System variable symbols
- Installability and usability enhancements
- Implementation improvements
- Compatibility and migration
- Special VSE/ESA feature

New and Enhanced Options

- New assembly-time options
  - ADATA
  - ASA
  - COMPAT
  - DXREF
  - EXIT
  - FOLD
  - LANGUAGE
  - LIBMAC
  - MXREF
  - OPTABLE
  - PCONTROL
  - PROFILE
  - RA2
  - SIZE
  - TRANSLATE
  - USING
  - XOBJECT
- New installation-time option
  - PESTOP
- Enhanced assembly-time options
  - FLAG
  - LINECOUNT
  - LIST
  - SYSPARM
  - TERM
  - XREF
- Source-file options
  - *PROCESS statement
- Old, unsupported options
  - ALGN
  - LINECNT=nn
  - LOAD
  - MULT
  - MSGLEVEL
- New default options

Listing Enhancements

- General
  - Print-line records 121 characters or longer
  - Two listing formats: narrow (121), wide (133)
  - Headings controlled by LANGUAGE option
- Options summary page
  - Displays invocation, *PROCESS, and actual options; overriding ddnames
- External symbol dictionary
  - XOBJECT extensions, ALIAS information
- Source and object code listing
  - USINGs-in-effect heading lines
  - LOC, C-LOC, D-LOC, R-LOC location counter headings
  - USING resolution details: registers, offsets
  - 'C' statement-number tag for COPYed statements,
    '-' for AREAD
  - Location counter displayed in PRINT NOGEN regions
  - Improved page-break handling
- Relocation dictionary
  - Wide-format extensions

Listing Enhancements ...

- Ordinary symbol and literal XREF enhancements
  - Relocation ID, relocatability tags, symbol type
  - Compacted cross reference: leading zeros suppressed
  - Branch, Drop, Modification, Using, eXecute tags
  - XREF(UNREFS) for unreferenced non-DSECT symbols
- Macro and COPY-member summary and XREF
  - Data set ID, COPY and LIBMAC tags, where defined, who called
  - Cross reference shows all uses
- DSECT cross reference
  - Relocation ID, length, definition-start statement
- USING map
  - Statement-location data, base address, type of USING, anchor location or registers, last-resolved statement, USING text
- Diagnostic and assembly summary page
  - Pointers to origins of source statements
  - Assembler name and fix level
  - All files used, I/O and exit counts
  - Start and stop times, estimate of processor time

Diagnostics Enhancements

- Improved diagnostic messages
  - Terminology clarified, better descriptions
  - Text insertions at more sensible positions
  - Messages in national languages (controlled by LANGUAGE option)
  - Severity-2 “Notifications” for low-impact conditions
- New diagnostics for new features
  - Options, initialization, exits, functions, statements
- New diagnostics for common problem areas
  - USING diagnostics
  - FLAG suboptions

ALIGN	check alignment
CONT	check continuations
RECORD	provide statement-origins detail
SUBSTR	check conditional-assembly substrings

- Note: Some warnings *may* apply to valid code
  - Easy “corrections” can eliminate such warnings
  - ***But:*** check first; don’t suppress the warnings!

Input-Output Exits

- I/O exits supported for all user files
  - Primary source (SYSIN)
  - Macro and COPY (SYSLIB)
  - Listing (SYSPRINT)
  - Punch (SYSPUNCH) and object (SYSLIN)
  - Terminal (SYSTERM)
  - Assembler data (SYSADATA)
- Integrate HLASM into development environments
- Full control over all I/O actions
  - Record insertion, deletion, modification
  - Cooperate with or replace assembler I/O
- Three sample exits are provided

  INPUT variable-format source records

  LISTING suppress or move options page

  ADATA call multiple record-selection and analysis routines

Assembler Data (SYSADATA) File

- A new programming interface for Assembler data
  - Eliminates need for listing scanners
  - Supports all analysis tools
- HLASM produces 19 record types, containing
  - Precise time stamp and character set ID
  - All option information
  - All source statements, with fields identified
  - All generated object code and object-file data
  - All symbols and attribute information
  - All cross reference data, including USING map
  - All messages
  - All files, members, and volumes referenced
  - All summary data
  - User-specified source-stream ADATA
- ASMADATA macro provides record mappings

Generalized Object File Format

- *Removes almost all limitations associated with old object module format*
  - External names to 63 characters
  - Section sizes up to 2GB (addresses to 31 bits)
  - Multicomponent, multimodal modules
    - One assembly can create many independently relocatable RMODE(24) and RMODE(31) segments
    - Entry points can have own AMODEs
  - Ability to retain assembler data (SYSADATA) with object code
  - And much more...
- Controlled by `XOBJECT` option
  - Cannot be specified with `DECK` or `OBJECT`
  - Requires wide (LIST(133)) listing format
- Utilizes new capabilities of DFSMS/MVS Binder

New Assembler Instruction Statements

- *PROCESS
  - Specify selected options in source module
- ADATA
  - Provide source-stream data for SYSADATA file
- AEJECT, ASPACE
  - Control format of macro-definition listings
- ALIAS
  - Specify alternatives to normal external symbols
- CATTR
  - Place object code in “Text Classes” (XOBJECT only)
- CEJECT
  - Control page EJECTs conditionally
- EXITCTL
  - Pass control information for exit routines
- RSECT
  - Check reentrancy on per-section basis

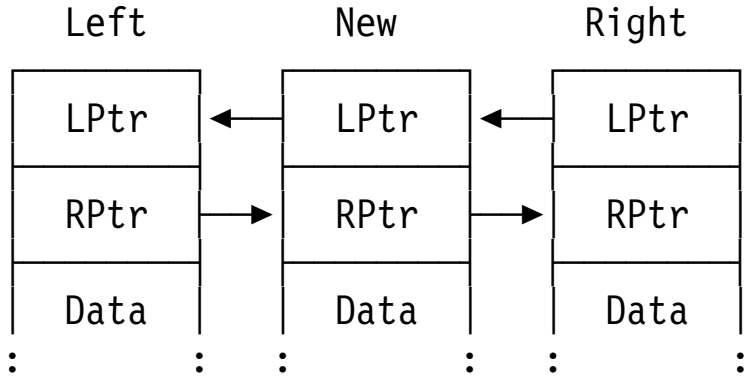
Enhanced Assembler Instruction Statements

- Two major extensions to USING statements: *labeled* and *dependent*
  - *Labeled* USINGs permit addressing multiple instances of a DSECT
  - *Dependent* USINGs permit addressing multiple DSECTs with a single base register
  - *Labeled dependent* USINGs combine their power!
- DROP extensions to support new USINGs
- COPY `&member` in open code
- Listing-control enhancements
  - New PRINT operands: [NO]MCALL, [NO]MSOURCE, and [NO]UHEAD
  - NOPRINT operand on PRINT, PUSH, POP
- AREAD operands for current time values

Labeled USINGs

- *Labeled* USINGs permit addressing multiple instances of a DSECT

  Example: Insert a `New` instance of `Block` in a doubly linked list between `Left` and `Right` elements




```
Block  DSect
LPtr   DS     A              Left sibling pointer
RPtr   DS     A              Right sibling pointer
Data   DS     XL273          Data area
```

Three instances of the DSECT named Block are concurrently active:

```
RNew   Equ    5              R5 points to NEW

Left   Using  Block,2        Labeled Using
Right  Using  Block,3        Labeled Using
New    Using  Block,RNew     Labeled Using

       MVC    New.LPtr,Right.LPtr  Link New to Right
       ST     RNew,Right.LPtr      Link Right to New
       MVC    New.RPtr,Left.RPtr   Link New to Left
       ST     RNew,Left.RPtr       Link Left to New
```

Dependent USINGs

- *Dependent* USINGs permit addressing multiple DSECTs with a single base register

  Example: Three distinct control blocks reside in adjacent areas of storage, anchored with a single register.

```
CB1    DSect ,          Define control block 1
CB1F1  DS    D
CB1F2  DS    CL40
LCB1   Equ   * CB1     Length of block 1

CB2    DSect ,          Define control block 2
CB2F1  DS    24F
LCB2   Equ   * CB2     Length of block 2

CB3    DSect ,          Define control block 3
CB3F1  DS    XL1000

*      Get storage for all 3 Blocks, address in R7

       Using CB1,7            Anchor full storage block

*                             Next 2 USINGs are Dependent
       Using CB2,CB1+LCB1     Adjoin CB2 to CB1
       Using CB3,CB2+LCB2     Adjoin CB3 to CB2

       STM   14,12,CB2F1+12   Addresses resolved with
       XC    CB3F1(4),CB3F1   ...a single base register
```

Labeled Dependent USINGs

- *Labeled dependent* USINGs combine their benefits

  Example: code and two DCB mappings addressed with a *single* base register

```
         Using *,12

InDCB    DCB   DDNAME=..., etc.
OutDCB   DCB   DDNAME=..., etc.

In       Using IHADCB,InDCB   Labeled dependent Using
Out      Using IHADCB,OutDCB  Labeled dependent Using

*        Following addresses are all resolved via R12
         MVC   Out.DCBLRECL,In.DCBLRECL

         DCBD  DSORG=PS       Generate IHADCB DSect
```

- Each instance of IHADCB DSect is “anchored” on a DCB
- Both DSects are active simultaneously
- All code and DSect addressing based on R12

USING Diagnostics

- USING in statement 3 nullifies USINGs in 2 and 4

```
                              1 START  CSect
        R:A  00000            2        Using *,10
        R:B  00000            3        Using *,11
ASMA301W ** WARNING ** Prior active USING on statement
                       number 2 overridden by this USING
        R:9  00000            4        Using *,9
ASMA300W ** WARNING ** USING overridden by a prior active
                       USING on statement number 2
```

- Message for statement 6 requested by specifying `USING(LIMIT(X'F00'))` option

```
 4120 BFFA          00FFA     6        LA    2,START+4090
ASMA304W ** WARNING ** Displacement exceeds limit value
                       specified
```

- Overlapping USING ranges

```
        R:7  00004            8        Using *,7
ASMA303W ** WARNING ** Multiple address resolutions may
                       result from this USING and the USING
                       on statement number 4
```

- R0 is only sometimes a valid base register!

```
                      00002    10 B      Equ   2
                      00000    11        Using B,0
ASMA302W ** WARNING ** USING specified Register 0 with a
                       nonzero absolute or relocatable
                       base address
```

New and Enhanced Language Elements

- Blank input records
- Mixed-case input
  - Controllable with COMPAT(CASE) option
- Underscore permitted as alphabetic
- Unary minus allowed in most expressions
- Literals allowed in more places

```
TR  NUM,=C¢0123456789ABCDEF¢ C¢0¢    Literal as a term
IC  0,=AL1(0,1,1,2,1,2,2,3,1,2)(R7)  Indexed literal
```

- Symbol attribute reference extensions and enhancements
  - Type, scale, integer attributes allowed in open code
- And many other niceties...

New/Enhanced Conditional-Assembly Language (1)

- Fourteen new internal conditional-assembly functions
  - Boolean XOR operator
  - SETA masking: AND, OR, XOR, and NOT
  - SETA shifts: SRA, SLA, SRL, SLL
  - Unary character functions
    - UPPER and LOWER: change “case” of letters
    - DOUBLE: pairs apostrophes and ampersands
  - Binary character functions
    - INDEX: finds first match of a string within another
    - FIND: locates first match of any character of one string within another
- External (user-supplied) functions
  - SETAF statement: invokes an integer-valued function
  - SETCF statement: invokes a character-valued function
  - Both types may have zero to many arguments
- Constructed lists may be passed as structures

```
     OUTERMAC A,(B,C,D),E     (B,C,D) (&P2) a list
                              OUTERMAC calls INNERMAC
     INNERMAC STUFF,&P2       Substituted &P2=¢(B,C,D)¢
* &P2 treated by INNERMAC as a string (COMPAT(SYSLIST))
*                        or as a list (NOCOMPAT(SYSLIST))
```

New/Enhanced Conditional-Assembly Language (2)

- New Opcode attribute reference, 0′
- New conditional-assembly substring notation ′&S′(n,*)
- Predefined absolute symbols in conditional-assembly expressions

```
ABS   EQU   20              Absolute predefined symbol
&VAR  SETA  10*ABS+4        SETA expression using ABS
      AIF   (&X GT ABS).A   AIF with predefined symbol
```

- Fewer restrictions on macro-call name field operand

```
75    MYMAC  TERMS,OPTIONS     Numeric name field
```

- Macro comment ′.*′ allowed in open code
- No & required on LCLx/GBLx declarations
- Many new system variable symbols
  - Assembly environment
  - Date and time
  - Option-dependent
  - Statement-oriented information
  - Statement-location information
  - Complete name/member/volume data for all files

System Variable Symbols

1. Assembly-environment variables
   - &SYSASM: Name of the Assembler
   - &SYSVER: Assembler version, release, and modification level
   - &SYSTEM_ID: Operating system under which this assembly is being performed
   - &SYSJOB, &SYSSTEP: Assembly job and step name
   - &SYSPARM: Assembler invocation parameter(*)
2. Date and time variables
   - &SYSDATC: Assembly date (format YYYYMMDD)
   - &SYSDATE: Assembly date (format MM/DD/YY)(*)
   - &SYSTIME: Assembly start time (format HH.MM)(*)

(*) Existed in Assembler H

System Variable Symbols ...

3. Option-dependent variables

   - &SYSOPT_DBCS: DBCS option setting
   - &SYSOPT_RENT: RENT option setting
   - &SYSOPT_OPTABLE: Name of the operation-code table used for the assembly; set by the OPTABLE option

4. Statement-related variables

   - &SYSSEQF: Contents of the sequence field of the current input statement
   - &SYSSTMT: Number of the next statement to be processed by the Assembler

5. Current statement-location variables

   - &SYSLOC: Name of current location counter(*)
   - &SYSSTYP: Type of the current control section into which statements are being grouped
   - &SYSNEST: Nesting level at which the current macro was invoked (macros called from open code are at level 1)

(*) Existed in Assembler H

6. File-information variables (three variables per file)

   - &SYS**xxx**_DSN: data set or file name
   - &SYS**xxx**_MEMBER: member name (if any)
   - &SYS**xxx**_VOLUME: volume identification

xxx	**Input Files**
IN	Current primary input file
LIB	Current library input file

xxx	**Output Files**
PRINT	Listing file
PUNCH	Object-module file
LIN	Object-module file
ADATA	SYSADATA file
TERM	Terminal-display file

Installability and Usability Enhancements

- Greater tailorability
  - Almost all options specifiable at invocation time
    - Options, messages, translation, opcode tables loaded dynamically
  - Individualized options possible
- Improvements to installation process
  - Extensions to install-time macros
  - Flexible installation choices, including aliasing to “old” product names
  - Uniform part names, with optional renaming steps
- All product publications extensively revised
  - New, improved (and more reliable!) examples
- Fewer manuals
  - General Information, Programmer’s Guide, Language Reference
  - One manual for installation, customization, diagnosis, and service information; eliminates need for Program Directories
- And much more...

Implementation Improvements

- Improved memory management
  - SIZE option controls storage allocation and use
  - Utility-file I/O used only when required
- Virtual storage constraint relief
  - Assembler code and data may reside above 16MB
    - One small I/O module must remain in 24-bit storage
  - Large storage reduces utility-file I/O
- Improved reliability and serviceability
  - Many internal enhancements and cleanups
  - New internal trace facility
  - Improved abnormal-termination information
- CMS interface module redesigned and rewritten
- Performance improvements
  - For large assemblies, fewer cycles than HLASM R1
  - QSAM used for all sequential nonutility files
  - System-determined block size (SDB) on MVS

Compatibility and Migration

- Upward compatibility for code that assembled correctly under Assembler H V2 and DOS/VSE Assembler
- COMPAT option to suppress new treatment of
  - Case sensitivity
  - Substituted macro sublists
  - Mixed-case macro operands
- Some previously undiagnosed situations now flagged
  - Options to control level and details of flagging
  - Recommend not disabling by default
  - New language elements
- More storage required, in general
  - HLASM is bigger: not a multiphase overlay structure
  - More information is collected and displayed
  - Table entries and macro dictionaries are larger
- *Many* extensions to the DOS/VSE Assembler
- New option defaults

Incompatibilities with Assembler H

1. Four new macro-time opcodes: ASPACE, AEJECT, SETAF, SETCF
    - Name conflicts will require additional handling
2. Five new assembly-time opcodes: ADATA, ALIAS, CATTR, CEJECT, EXITCTL
    - Name conflicts can be handled with OPSYN or COPY
3. Many new system (&SYS) variable symbols
    - Conflicts should **not** occur (symbols begin with &SYS, which was always reserved for the assemblers)
4. Obscure errors have been corrected
    - Type attributes of declared uninitialized variable symbols
        - HLASM returns 0 for SETC symbols, N for SETA and SETB
        - Assembler H returned 00 for SETA, U (with a diagnostic) for SETB, and 0 for SETC
        - Assembler XF returned N, N, and U, respectively
    - Type attribute of CNOP label is I, rather than J
    - Type attribute of literals is “reasonable” (not U)
5. Some previously unnoticed conditions now flagged
    - USING-range conditions, continuation ambiguities
        - Options to control level and detail of flagging
    - Apparent attribute references in open code

Enhanced Assembler Technology

- Maximizes productivity of critical skills
- Extensive support for application development, management, and maintenance

New Productivity and Reliability Features

- Enhanced language and functional features
- Extensive usability and adaptability enhancements
- New, expanded diagnostic and cross reference capabilities

Cost Savings

- Faster development cycles
- Increased application maintainability and reliability
- Greater programmer efficiency and productivity

Continuing IBM's commitment to support Assembler Language applications

Glossary

Assembler. A program that converts source statements written in *Assembler Language* into a *machine language object file* and provides additional useful information such as diagnostic messages, a formatted listing of the source and object code, and symbol usage cross references.

Assembler Language. The symbolic machine-oriented source language accepted by High Level Assembler.

machine language. The binary instructions and data interpreted and manipulated by the processor when the program is executed. It is not meant to be intelligible to human beings. Compare *Assembler Language*.

object file. A file, produced by the *Assembler*, that contains the machine language representation of the assembled program.

option. A directive to the *Assembler* specifying various “global” controls over its behavior. For example, the OBJECT option specifies that the Assembler should produce an *object file*. The user specifies options as a string of characters, usually as part of the command or statement that invokes the Assembler.

Index

Special Characters

A

B

C

D

L

M

N

O

P

Q

R

S

T

U

V

X

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
High Level Assembler:

Technical Overview
December 1995

Publication No. SG24-3910-01

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 471/E2
650 Harry Road
San Jose, CA
USA 95120-6099

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.