High Level Assembler:

Toolkit Feature Technical Overview

SHARE 96 (February 2001), Session 8166

February, 2001

John R. Ehrman
ehrman@us.ibm.com

International Business Machines Corporation
Silicon Valley (Santa Teresa) Laboratory
555 Bailey Avenue
San Jose, California 95141

Note

Demonstration and trial versions of some Toolkit components are on the HLASM web site:

`http://www.ibm.com/software/ad/hlasm/`

Synopsis:

The examples in this document are for purposes of illustration only, and no warranty of correctness or applicability is implied or expressed.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

IBM	ESA
MVS/ESA	System/370
System/370/390	System/390
VM/ESA	VSE/ESA
VSE	OS/390

Publications, Collection Kits, Web Sites

The currently available product publications for High Level Assembler for MVS & VM & VSE are:

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

- High Level Assembler for MVS & VM & VSE *Toolkit Feature Interactive Debug Facility User's Guide*, GC26-8709
- High Level Assembler for MVS & VM & VSE *Toolkit Feature User's Guide*, GC26-8710
- High Level Assembler for MVS & VM & VSE *Toolkit Feature Installation and Customization Guide*, GC26-8711
- High Level Assembler for MVS & VM & VSE *Toolkit Feature Interactive Debug Facility Reference Summary*, GC26-8712

- High Level Assembler for MVS & VM & VSE *Release 2 Presentation Guide*, SG24-3910

Soft-copy High Level Assembler for MVS & VM & VSE publications are available on the following *IBM Online Library Omnibus Edition* Compact Disks:

- *VSE Collection*, SK2T-0060
- *MVS Collection*, SK2T-0710
- *Transaction Processing and Data Collection*, SK2T-0730
- *VM Collection*, SK2T-2067
- *OS/390 Collection*, SK2T-6700 (BookManager), SK2T-6718 (PDF)

HLASM publications are available online at the HLASM web site:

`http://www.ibm.com/software/ad/hlasm/`

IBM High Level Assembler: Toolkit Feature Overview

IBM High Level Assembler Release 4:
Toolkit Feature Technical Overview

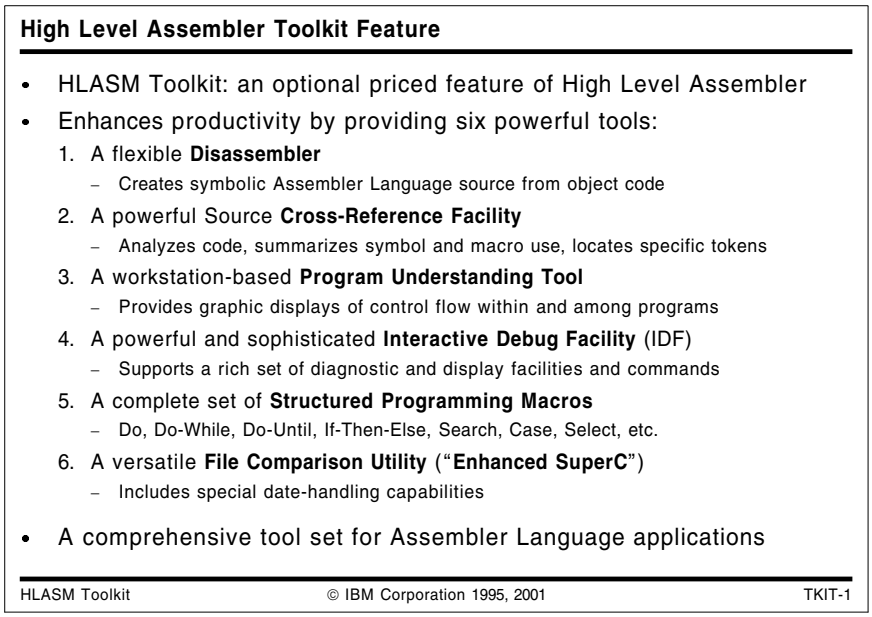
SHARE 96 (Feb. 2001), Session 8166

John R. Ehrman

IBM Silicon Valley (Santa Teresa) Laboratory
555 Bailey Avenue
San Jose, CA 95141
EHRMAN@VNET.IBM.COM

March 1, 2001

Trademarks and Copyright Notices



High Level Assembler Toolkit Feature

The High Level Assembler Toolkit Feature is an optional, separately priced feature of IBM High Level Assembler (HLASM). It provides a powerful and flexible set of six tools to improve application recovery and development, and to assist in program preparation, analysis, debugging, and maintenance on OS/390*, MVS/ESA*, VM/ESA*, and VSE/ESA* systems. These productivity-enhancing tools are:

- **Disassembler**, a tool which converts binary machine language to Assembler Language source statements. It helps you understand programs in executable or object format, and enables recovery of lost source code.
- **Cross-Reference Facility**, a flexible source-code analysis and cross-referencing tool. It helps you determine variable and macro usage, analyze high-level control flows, and locates specific uses of arbitrary strings of characters.
- **Program Understanding Tool**, a workstation-based program analysis tool. It provides multiple and “variable-magnification” views of control flows within single programs or across entire application modules.
- **Interactive Debug Facility**, a powerful and sophisticated symbolic debugger for applications written in Assembler Language and other compiled languages. It simplifies and speeds the development of correct and reliable applications. (It is not intended for debugging privileged or supervisor-state code.)
- **Structured Programming Macros**, a complete set of macro instructions that implement the most widely used structured-programming constructs (IF, DO, CASE, SEARCH, SELECT). These macros simplify coding and help eliminate errors in writing branch instructions.
- **File Comparison Utility** (known as “Enhanced SuperC”), a versatile file searching and comparison tool. It can scan or compare single file or groups of files with an extensive set of selection and rejection criteria. Typical uses include comparing an original source file with a modified source file, or a pre-migration application output file with a post-migration output file. Newly added functions include “smart comparisons” of date fields to assist Y2K migration and date “windowing.”

Together, these tools provide a powerful set of capabilities to speed application development, diagnosis, and recovery.

This presentation provides an overview of the features and use of each of the six Toolkit components. They are based on tools that have been used widely and tested extensively inside IBM before being “packaged” in the High Level Assembler Toolkit.

Hardware Requirements

The High Level Assembler Toolkit Feature requires the same hardware environments as IBM High Level Assembler for MVS & VM & VSE Version 1 Release 3. Requirements for 24-bit Virtual Storage are:

- Disassembler: 100K bytes
- IDF: 600K bytes
- XREF: depends on number and sizes of modules being scanned
- SuperC: depends on number and sizes of modules being scanned
- ...plus working storage (depending on the application)

The Program Understanding Tool (ASMPUT) component of the High Level Assembler Toolkit Feature requires a workstation capable of running OS/2, Windows 95, or Windows NT with a minimum of 16 MB memory (32 MB recommended) and 80 MB of available hard-drive space, plus a host-sytem connection or other means of transferring SYSADATA files to the workstation for analysis.

Software Requirements

The High Level Assembler Toolkit Feature operates in all MVS/ESA and VM/ESA environments where IBM High Level Assembler for MVS & VM & VSE Version 1 Release 3 (MVS & VM Edition) operates. On MVS, the Interactive Debug Facility's macro facilities require TSO/E V2 or later.

On OS/390, the High Level Assembler Toolkit Feature is an optional element; it operates in all environments where the same level of the High Level Assembler base element operates.

The High Level Assembler Toolkit Feature operates in VSE/ESA Version 2 (or later) environments where IBM High Level Assembler for MVS & VM & VSE Version 1 Release 3 (VSE Edition) operates. On VSE, the Interactive Debug Facility requires VSE Version 2.2 or later.

The Toolkit Feature's components can be used independently of HLASM. However, the most productive uses of most of the Toolkit Feature's components rely on SYSADATA files produced by High Level Assembler.

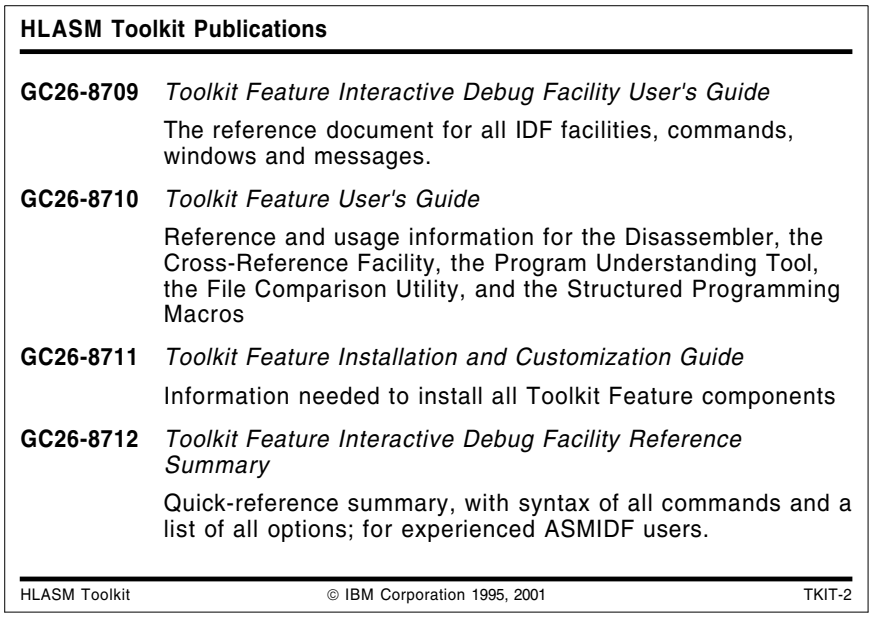
Note: The SYSADATA files should not be created if the GOFF or XOBJECT option is in effect.

The Program Understanding Tool (ASMPUT) component of the High Level Assembler Toolkit Feature requires APAR PQ26063 and one of:

- OS/2 Version 4 (8H1425) with fixpack 8 or later
- Windows 95
- Windows 98
- Windows NT Version 4.0 with Service Pack 3 or later, on Intel workstations only.

A recommended host-connection software package is eNetwork Personal Communications Version 4.2.1 (8H8735), which supports OS/2 and Windows.

Note: The OS/2 version of ASMPUT shipped with HLASM R3 works only with HLASM R3 ADATA files. If you have ADATA files generated by HLASM R2 and want to continue to use them, you should retain your copy of the OS/2-based ASMPUT shipped with HLASM R2.



Publications

The four publications for the High Level Assembler Toolkit Feature are:

GC26-8709 *Toolkit Feature Interactive Debug Facility User's Guide*

The main reference document that describes all IDF facilities, commands, windows and messages.

GC26-8710 *Toolkit Feature User's Guide*

Reference and usage information for the Disassembler, the Cross-Reference Facility, the Program Understanding Tool, the Enhanced SuperC File Comparison Utility, and the Structured Programming Macros

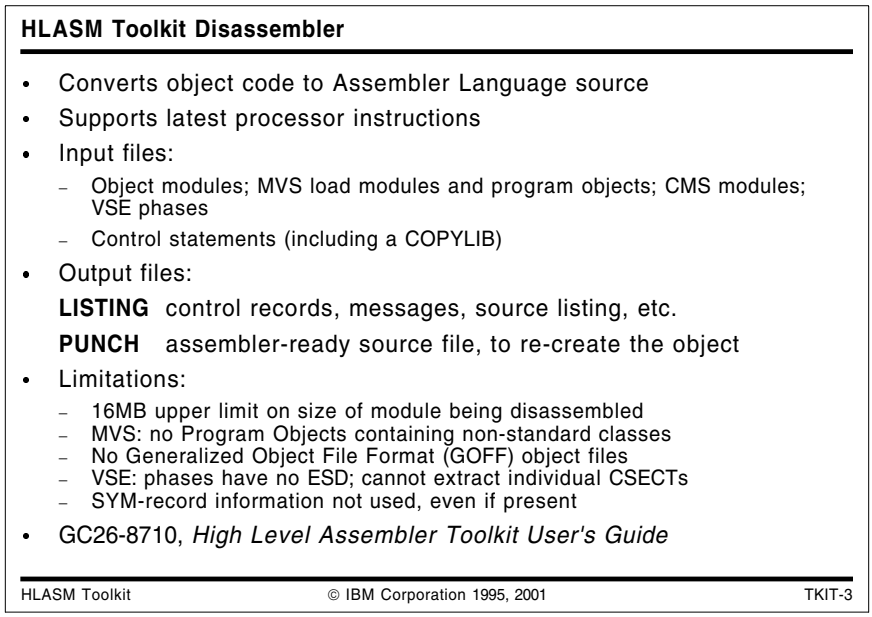
GC26-8711 *Toolkit Feature Installation and Customization Guide*

Information needed to install all Toolkit Feature components

GC26-8712 *Toolkit Feature Interactive Debug Facility Reference Summary*

Quick-reference summary, with syntax for all commands and a list of all the options. This booklet is intended for experienced ASMIDF users.

For more information about ordering the High Level Assembler Toolkit Feature, refer to Software Announcement 295-498, dated December 12, 1995.



HLASM Toolkit Disassembler

The HLASM Toolkit Feature's Disassembler lets you extract single control sections (CSECTs) from object modules or from executables such as MVS load modules, CMS modules, and VSE phases. It converts them to Assembler Language statements that can be assembled to generate the same object code. A control file (including a COPYLIB of previously created control statements) supplies information to guide the Disassembler in producing a more readable and modifiable output source program.

The Disassembler produces two output files:

Listing Various sections describe the module being disassembled, control records, messages, text listing and the source listing.

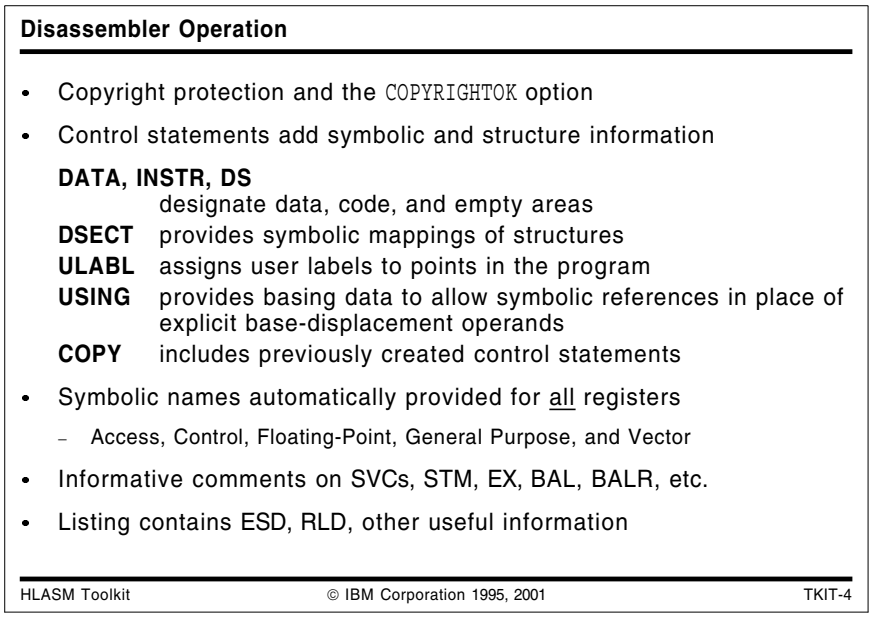
Punch An assembler language source file that can be used directly as input to the assembler to recreate the object text file.

The Disassembler currently has the following limitations:

- 16MB upper limit on the size of the module being disassembled
- On MVS: Program Objects containing non-standard classes (i.e., classes not defined and owned by the DFSMS/MVS Binder) cannot be disassembled.
- Generalized Object File Format (GOFF) object files cannot be disassembled.
- On VSE: Because VSE executable phases have no External Symbol Dictionary (ESD), the Disassembler cannot extract individual CSECTs, nor produce a useful ESD report.

  Note: VSE utilities can create an object-module file from a phase; that object module may also be disassembled.
- SYM-record information is not used, even if present in the object file or load module.

Publication GC26-8710, *High Level Assembler for MVS & VM & VSE Toolkit Feature User's Guide, MVS & VM* describes all the control records, JCL requirements, and error messages for the Disassembler.



Disassembler Operation

The COPYRIGHTOK option controls the processing of control sections that contain copyright information. By default, the disassembler will scan the object code for the following data:

- (c)
- (C)
- © (at code point X'B4')
- “Copyright” in any combination of upper case and lower case letters.

If any one of these is found, message ASMD010 will be issued and the disassembly will stop. However, if you specify the COPYRIGHTOK option, then you are acknowledging that you own the copyright for the module or that you have obtained permission from the copyright owner to disassemble the module. In this case the Disassembler will issue message ASMD008 to acknowledge this, and processing will continue.

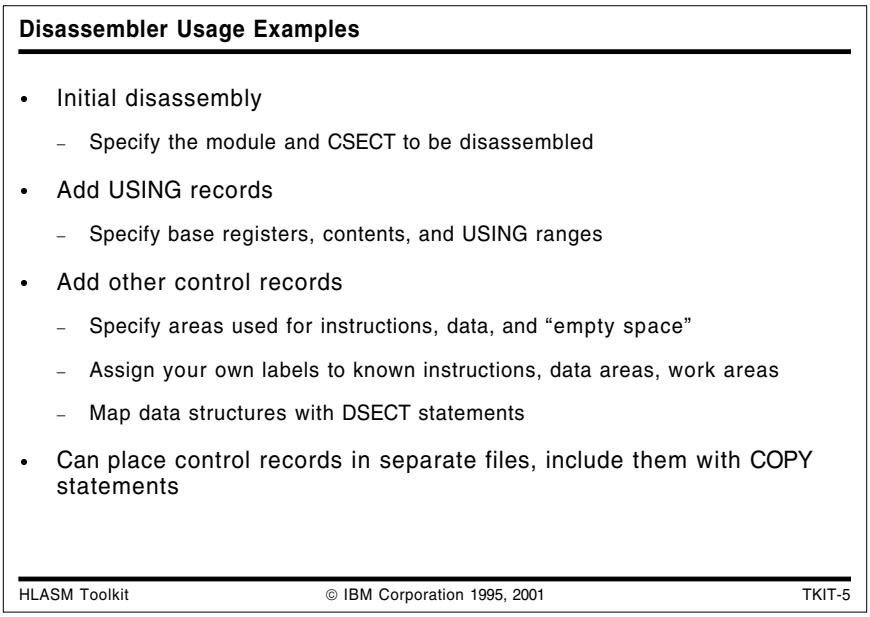
The Disassembler operates in two passes: Pass 1 reads and processes all the control records, and builds storage tables for later use. The main tables are for labels, USINGs and DSECTs. Pass 2 performs the actual disassembly, analyzing the module's machine language text and writing assembler language instructions to the listing and punch files.

Your first control statement specifies the module and control section to be disassembled. Additional control statements provide further guidance and helpful information to the Disassembler, allowing it to create a more readable program. You can supply sets of control statements in the primary input stream to the Disassembler, or (as each set is developed) you can save them in a library and direct the Disassembler to read them using COPY control statements.

- You can describe the layout of the control section with control statements asserting that certain areas of the module contain data only, instructions only, or are known to be uninitialized.
- You can request symbolic resolutions of halfword base-displacement storage by supplying control statements giving base addresses and the base registers to be used for addressing.
- You can assign your own labels to designated positions in the program, and define data structures (DSECTs).

- The Disassembler automatically assigns symbolic names to registers. Branch instructions use extended mnemonics where possible, and supervisor call (SVC) instructions are identified when known. (The Disassembler cannot create source programs that recover original macro calls, of course!)
- The Disassembler listing provides a full summary of the inputs and outputs of the disassembly, and the reconstructed Assembler Language source program is placed in a separate PUNCH file.

When the disassembler-generated statements are assembled by High Level Assembler using the ADATA option, the resulting SYSADATA file (also called the ADATA file) may be used as input to other Toolkit Feature components. This combination of facilities can help you recover lost source code written in *any* compiled language.



Disassembler Usage Examples

Some examples of the disassembler will use the object file from the program listed in Figure 1 on page 8 below. The object text file in each of the following three examples is identical. Each example has its own set of control records:

1. Initial run (DISASM1)
2. Add USING records (DISASM2)
3. Add other control records (DISASM3)

The control files used for these disassemblies are discussed starting at “Sample Disassembler Control Files: DISASM1” on page 8. (Note that because the examples were run under CMS, the first operand of the first control statement is ignored; the name is used only to distinguish the three samples.)

```
Trace CSect
      stm   r14,r12,12(r13)         Save caller's registers
      lr    r12,r15                 Establish Base
      Using trace,r12               and tell the assembler
      st    r13,savearea+4          Chain
      la    r2,savearea               the
      st    r2,8(,r13)                 saveareas
      la    r15,12                  set default return code
      oc    myflag,myflag           Have we been here?
      bz    exit                    Get out now
      xc    myflag,myflag           Clear the flag
      st    r8,areaaddr             Set up address of area
      mvc   arealen,=f'8192'        Set up length of area
      la    r15,8                   Set the return code

exit  l     r13,savearea+4          Point to previous area
      st    r15,16(,r13)            Store the return code
      mvc   24(4,r13),=a(dump_data) Point saved R1 at Parm list
      lm    r14,r12,12(r13)         Restore the registers
      br    r14                     Return
      Ltorg
myflag    dc  F'1'
savearea  ds  9d
dump_data dc  f'1'
areaaddr  dc  a(0)
arealen   dc  f'0'
titlea    dc  a(title)
title     dc  cl16'Hello world'
r2        Equ 2
r8        Equ 8
r12       Equ 12
r13       Equ 13
r14       Equ 14
r15       Equ 15
      End
```

Figure 1. Sample Program for Disassembly

Sample Disassembler Control Files: DISASM1

The initial version of the control file specifies only a single statement, to designate the module name and the CSECT name. (Under CMS, the module name DISASM1 is ignored.)

```
DISASM1 TRACE
* This is the minimum requirement - the control record which
* specifies the module (not used on VM) and the CSECT.
```

The output of this disassembly contains no USING statements and no internal labels; all addressing is in base-displacement form, as illustrated in Figure 2 on page 9 below. (Note that the last statement before END is a call on the ASMDREG macro: this macro is supplied with the Toolkit Feature, and simply defines the names of the general purpose registers R0 through R15. It is equivalent to the REGEQU macro, which unfortunately is not available on all platforms.)

```
TRAC     TITLE 'Disassembly of CSECT TRACE    of Load Module DISASM1 '
*               Produced by ASMDASM on 98.120 at 14:06
TRACE    CSECT
         STM   R14,R12,12(R13)             Save regs
         LR    R12,R15
         ST    R13,92(,R12)
         LA    R2,88(,R12)
         ST    R2,8(,R13)
         LA    R15,12
         OC    80(4,R12),80(R12)
         BZ    52(,R12)
         - - -   ...etc...
         DC    CL16'Hello world     '
         ASMDREG
         END
```

Figure 2. Sample Disassembly With Minimal Control Statements

Sample Disassembler Control Files: DISASM2

After inspecting the initial disassembly, we have determined that register 12 should be used as a base register. (Again: under CMS, the module name DISASM2 is ignored.)

```
DISASM2 TRACE
* Now we have added a USING record which specifies that
* the USING applies to all addresses between X'000000' and X'0000C0',
* register 12 (X'C') is to be used as a Program base register
* and that the value loaded into the register is X'000000'
USING 000000 0000C0 C P 000000
```

The output from this disassembly would use symbolic labels for storage references based on register 12. The generated names are of the form Annnnnn where nnnnnn is the hexadecimal offset of the label from the base of the control section. This is illustrated in Figure 3 below.

```
TRAC     TITLE 'Disassembly of CSECT TRACE    of Load Module DISASM1 '
*               Produced by ASMDASM on 96.176 at 14:32
TRACE    CSECT
         USING *,R12
A000000  EQU   *
         STM   R14,R12,12(R13)             Save regs
         LR    R12,R15
         ST    R13,A00005C
         LA    R2,A000058
         ST    R2,8(,R13)
         LA    R15,12
         OC    A000050(4),A000050
         BZ    A000034
         XC    A000050(4),A000050
         - - -   ...etc...
A0000B0  EQU   *
         DC    CL16'Hello world     '
         ASMDREG
         END
```

Figure 3. Sample Disassembly With USING Control Statement

Sample Disassembler Control Files: DISASM3

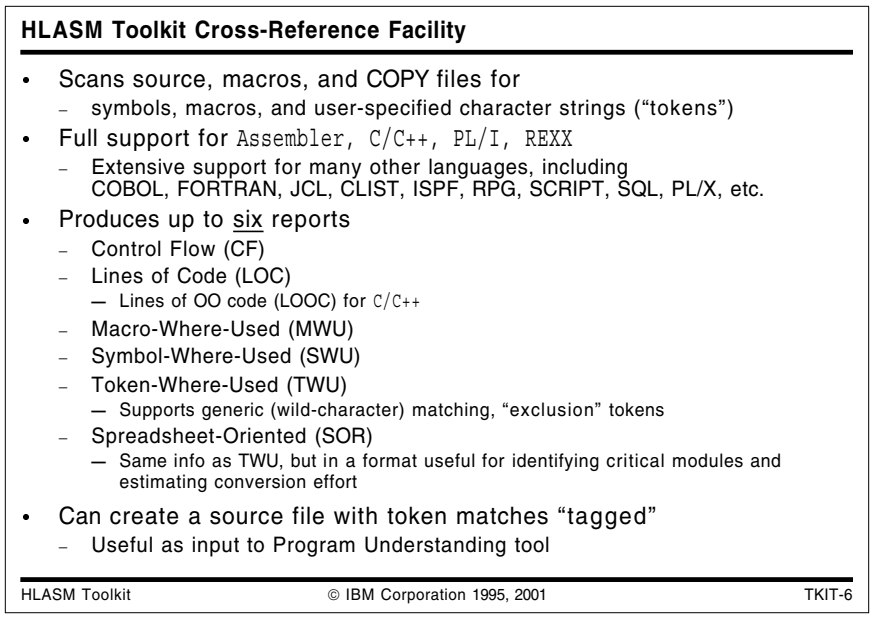
For the final disassembly, we observe that there is a save area at offset X'000058' that we will call SAVEAREA, and this area is uninitialized space; also, there appears to be a fullword at offset X'000050' used as a FLAG.

```
DISASM3 TRACE
USING 000000 0000C0 C P 000000
* The following defines a label SAVEAREA for an area which starts at
* offset X'000058' and is 72 bytes long (18 fullwords)
ULABL SAVEAREA 000058 072
* This defines the area from X'000058' to X'00009F' as an
* uninitialized storage area (this will force the use of the DS opcode)
DS 000058 00009F
* another label definition - FLAG at offset X'50' for 4 bytes
ULABL FLAG     000050 004
```

The output from this (possibly final) disassembly is shown in Figure 4 below:

```
TRAC     TITLE 'Disassembly of CSECT TRACE    of Load Module DISASM1 '
*               Produced by ASMDASM on 1998.120 at 17:58
TRACE    CSECT
         USING *,R12
A000000  EQU   *
         STM   R14,R12,12(R13)             Save regs
         LR    R12,R15
         ST    R13,SAVEAREA+4
         LA    R2,SAVEAREA
         ST    R2,8(,R13)
         LA    R15,12
         OC    FLAG(4),FLAG
         BZ    A000034
         XC    FLAG(4),FLAG
         ST    R8,A0000A4
         MVC   A0000A8(4),A000048
         LA    R15,8
A000034  L     R13,SAVEAREA+4
         ST    R15,16(,R13)
         MVC   24(4,R13),A00004C
         LM    R14,R12,12(R13)             Restore regs
         BR    R14                         Exit
         SPACE
A000048  DC    F'08192'
A00004C  DC    A(A0000A0)
FLAG     DC    F'00001'
         DC    F'0'
SAVEAREA DS    CL72
A0000A0  EQU   *
         DC    F'00001'
A0000A4  DC    F'0'
A0000A8  DC    F'0'
         DC    A(A0000B0)
A0000B0  EQU   *
         DC    CL16'Hello world     '
         ASMDREG
         END
```

Figure 4. Disassembler Output for Sample Program



HLASM Toolkit Cross-Reference Facility

The High Level Assembler Toolkit cross-reference tool (ASMXREF) supports your maintenance tasks by analyzing and scanning source programs, macro definitions, INCLUDE and COPY books and other files for symbols, macro calls, and user-specified tokens. The source programs may be written in Assembler Language, C/C++, PL/I, or REXX. Other languages supported for a subset of the available reports include COBOL, FORTRAN, ASM88, CLIST, “Generic,” ISPF panels and skeletons, JCL, MASM, Modula, Pascal, QMF/SQL, RPG, and SCRIPT.

ASMXREF can also be used for identifying fields of application importance such as DATE, TIME, and YYMMDD. You additionally specify tokens to be *excluded*, so that searches for a token such as "MM" can reject matches on tokens such as SUMMER. Furthermore, an arbitrary “match anything” character (sometimes called a wildcard character) can be used to create generic tokens such as "YY*"; the scan will then search for occurrences of the token with any other characters allowed in the position of the arbitrary character.

ASMXREF does not support VSAM files.

Control Flow Report

The CF report tabulates all intermodule program references as a function of member or entry point name. Additional language-specific capabilities are provided for selected languages.

Lines of Code Report

The LOC report provides a count, arranged by part, of the number of source lines in the part, the executable and non-executable statements and the number of comment lines in the part. Appropriate tags can be used to indicate lines changed, deleted, added, or moved, as well as to indicate programmer activity.

Lines of Object Oriented Code Report

There is a special subset of the LOC report for C/C++: the LOOC (Lines of Object Oriented Code) reports the Lines of Code (LOC) per class and per object, and objects per class, containing data similar to that in the “standard” LOC report. “Shipped Source Instructions” (SSI) indicates the number of executable and non-executable instructions that are not blank or comments.

Macro Where Used Report

The MWU report lists all macros or functions invoked and all segments copied, including the type and frequency of the invocation or reference.

Symbol Where Used Report

The SWU report lists all symbols referenced within the source members, and the type of reference. These symbols can be variables or macros.

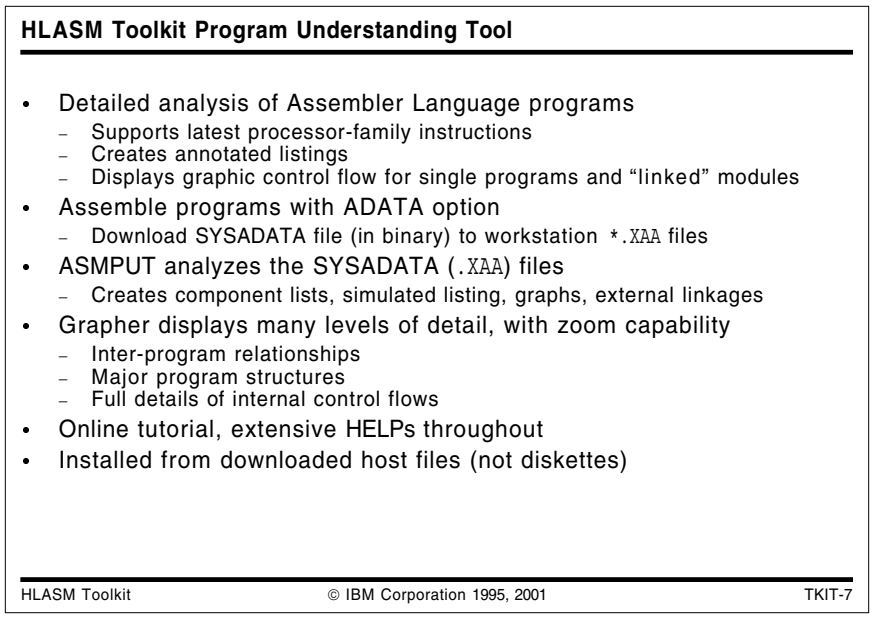
Token Where Used Report

The TWU report shows for each module scanned the number of lines of code, the number of occurrences of each token, and the total number of token matches. Tokens may also be excluded from matching.

When you create the TWU report, a “tagged source program” is also generated. This file contains special language-specific inserted comment statements where tokens are found. Subsequent assembly of a “tagged” file helps you track important variables during control-flow analysis using the Program Understanding Tool.

Spreadsheet Oriented Report

The SOR report contains the same information as the TWU report, as a comma-delimited file suitable for input into a standard spreadsheet application. This tabular information helps you identify the critical modules in an application and estimate the effort required for needed modifications.



HLASM Toolkit Program Understanding Tool

The Program Understanding Tool (ASMPUT) helps you analyze and extract information about Assembler Language applications, using a graphical user interface to display graphical and textual views of an application's structure. ASMPUT extracts application analysis information from the SYSADATA file generated during host assembly by HLASM; this ADATA file is downloaded to the workstation for analysis. **Note**: The High Level Assembler Toolkit Feature ASMPUT requires ADATA files generated by the same release of HLASM.

ASMPUT can display linked views of selected programs and modules including:

- a Content view
- an Assembled Listing view
- a graphical Control Flow view
- an Expanded Source Code view.

These views provide complete high, medium, and low level information about Assembler Language applications.

- At the highest level, you can discover the relationships among programs and modules within an application.
- A mid-level view displays the calling structures among programs within a module, including routines external to a program.
- At the lowest level, you can examine details of internal control flows within each program.

ASMPUT lets you display multiple views of a given program or module. These multiple views are linked: scrolling through one view automatically scrolls through all other open views of that program, module, or application. Linked views help you see quickly the association between the assembled source code and the graphical control-flow representations of the program.

At any time, you can narrow or expand the focus of your analysis by zooming in or out on areas of particular interest. For example, you can use the VIEW CONTENTS window to scroll through the contents of an application and simultaneously see the change in control flow information displayed in the VIEW CONTROL FLOW window.

ASMPUT displays several folders which provide a complete inventory of application analysis information, program samples, tools, documentation, extensive help files, and a detailed online tutorial to help you learn to use ASMPUT for analyzing Assembler Language applications. Installation is simplified by packaging all Toolkit components as host files; ASMPUT files are then downloaded to the workstation.

The initial window gives direct access to all needed files, functions, and information needed to analyze assembler language programs.

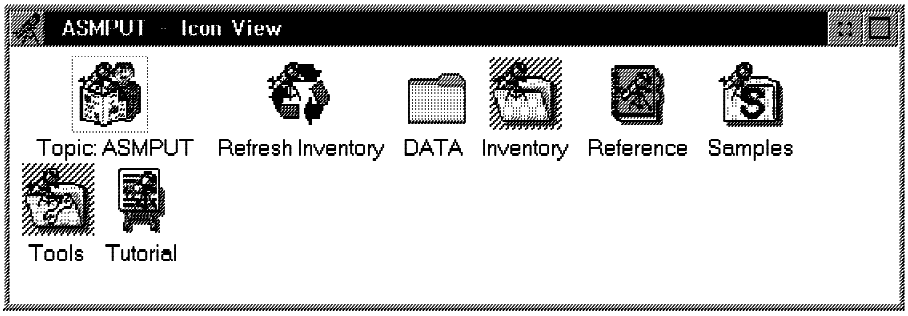


Figure 5. Example of the ASMPUT window

The shaded icons in this window indicate that the **Tools** and **Inventory** windows are also open. Following ADATA analysis, you can display many different views of a program. A view of the initial analysis might be the source file, as shown in the following figure:

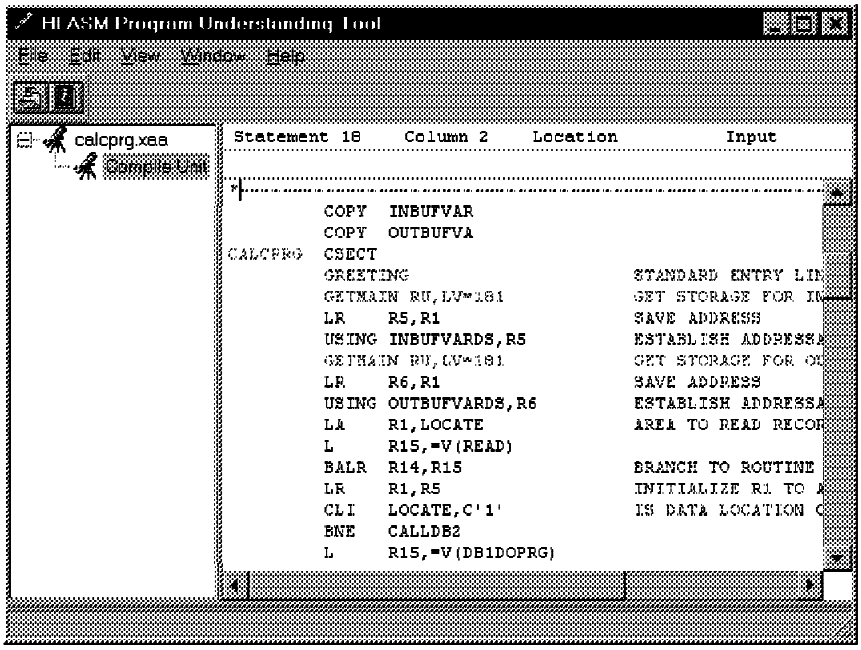


Figure 6. Example of the ASMPUT source listing

The Program Understanding Tool uses different colors to highlight machine, assembler, and macro instructions. Other listings display the program's components (source, macro, and COPY files), or the control flow analysis, where “basic blocks” (sequences of instructions ending at a branch) are identified.

The control flow graphs are the heart of ASMPUT. For example, a top-level view of the control flow graph for the CALCPRG sample program appears like this:

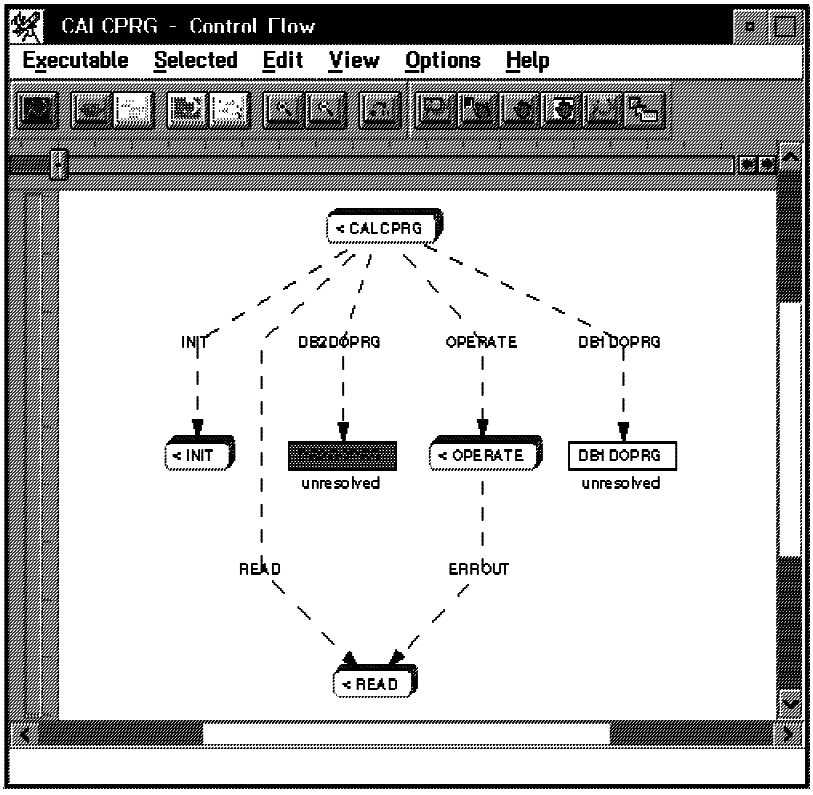


Figure 7. Example of the Control Flow View

The next level of detail shows the structure of each of the routines called from the main CALCPRG program:

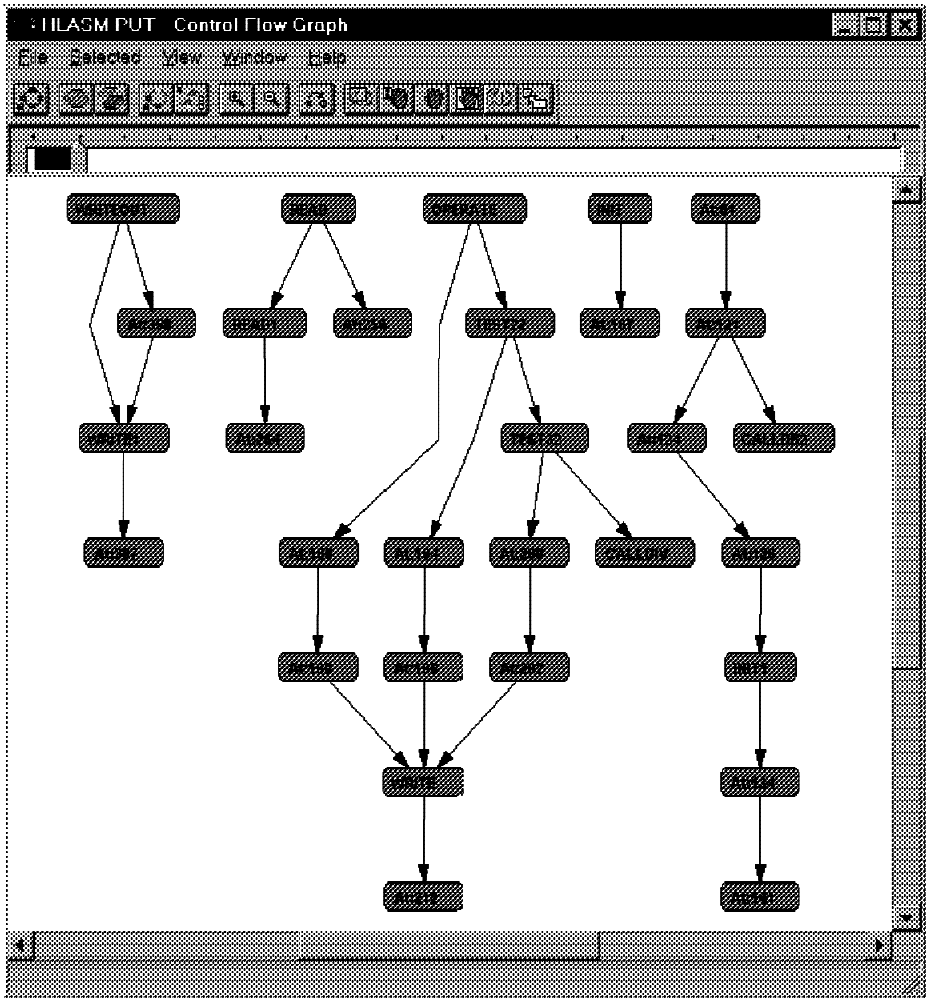
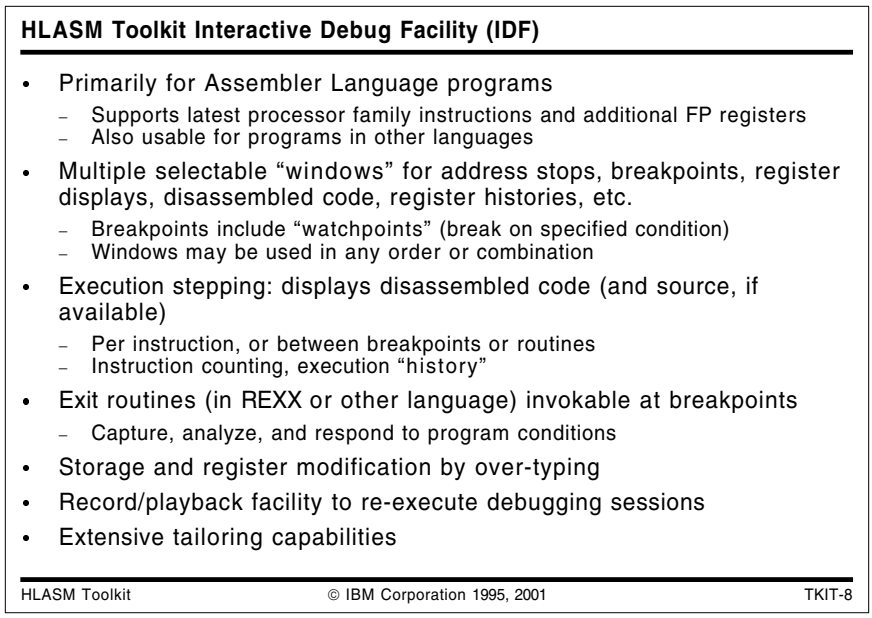


Figure 8. Example of a More Detailed Control Flow View

Using the **View** pull-down, you can expand or collapse the “layers” of detail being displayed.

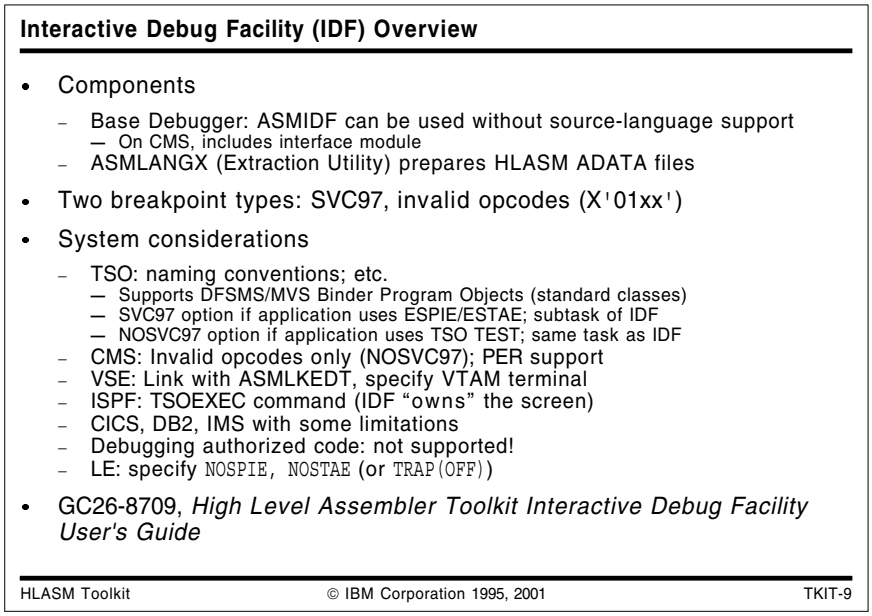


HLASM Toolkit Interactive Debug Facility (IDF)

The HLASM Toolkit Feature Interactive Debug Facility (IDF) supports a rich set of capabilities that speed error detection and correction. While IDF is intended primarily for debugging Assembler Language programs on MVS, VM, and VSE systems, it can also be used advantageously to debug programs written in most high level languages.

- IDF supports all new z/Architecture instructions and the additional floating-point registers introduced with the G5 processor families. (It also shares a common disassembly routine with the Disassembler and several other system components, ensuring correct handling of all instructions by each.)
- IDF provides multiple selectable views of a program, including separate windows for address stops, breakpoints, register displays, object code disassembly, storage dumps, language-specific support, register histories, non-traced routines, and other information. These views can be used in any order or combination.
- Execution of a program can be controlled by stepping through individual instructions or between selected breakpoints or routines.
- If source code is available (which will almost always be the case for programs assembled with HLASM), IDF can display source statements as the program is executed.
- The power of IDF is greatly magnified by its ability to pass control from any breakpoint to user exit routines written in REXX or other languages that can capture and analyze program data, and respond dynamically to program conditions.
- Instruction executions can be counted, and an instruction execution history can be maintained.
- Storage areas and register contents can be modified dynamically during debugging by simply typing new values on the displays.
- IDF supports a special class of conditional breakpoints called watchpoints, which are triggered only when a user-specified condition occurs.
- A command-level record and playback facility allows a debugging session to be re-executed automatically.

- Extensive tailoring capabilities allow you to establish a familiar debugging environment. Most debugging actions can be easily controlled by PF-key settings.



Interactive Debug Facility (IDF) Overview

The original IDF provided a debugger without any source-level capability. It can still be used in that way, and any reference to the "base debugger" implies using IDF without its source language capabilities.

IDF comprises two main components:

1. On TSO, the base debugger is the load module ASMIDF. This is a TSO command processor; it will only run in that environment with a real terminal.

   On CMS, the base debugger consists of two modules; ASMIDF and ASMIDFMA. ASMIDF is a self-relocating nucleus extension. This loads the main module, ASMIDFMA, as a nucleus extension at the start of a debugging session, and deletes it at the end.

   On VSE, the ASMIDF debugger runs in batch mode. A VTAM terminal must be available.

2. The other component of ASMIDF is ASMLANGX, the extraction utility that reads SYSADATA files and creates ASMLANGX files for later use by the language-support component of ASMIDF. The ASMLANGX utility can process SYSADATA files created by either HLASM Release 2 or Release 3.

IDF uses two different breakpoint techniques, both of which overlay instructions at the point where the breakpoint is required:

TSO Invalid opcodes of the form X'01xx' or `SVC 97` instructions

CMS Invalid opcodes of the form X'01xx' (`SVC 97` not supported)

VSE Invalid opcodes of the form X'01xx'

The implications of these choices will be described shortly. IDF inserts these breakpoint opcodes when it is about to begin executing the target program. When any "event" occurs, the original instructions are restored before control is returned to you, so that all displays will show your program without the breakpoint overlays. Note that some other debuggers depend on having the compiler insert special links to the debugger, which limits their

usefulness for code that is fully optimized for production environments. IDF, on the other hand, is a lower-level debugger that uses opcode overlays to set breakpoints.

- TSO considerations
  - Debuggable modules

    IDF supports debugging of programs in both the old load module format and in the new Program Object format produced by the DFSMS/MVS Binder (for PM1, PM2 and PM3) so long as the Program Object classes are those assigned and owned by the Binder.

  - SVC97 and NOSVC97 options

    By default, ASMIDF uses the TSO TEST SVC (SVC97). You must use the SVC97 technique when debugging an application which itself uses ESTAE or ESPIE. This is because the application's ESTAE/ESPIE setup will take precedence over IDF's. (This is not available under ISPF unless you use the standard TSOEXEC command to set up the appropriate environment; the same restriction applies to the TSO TEST command.)

    NOSCV97 works by telling ASMIDF that it is not to use SVC97; it then switches to using invalid opcodes to set breakpoints.

  - TSO naming conventions

    ASMIDF was originally developed as a CMS tool and later ported to TSO, so there are a lot of CMS conventions throughout the manuals. TSO users must translate their DDNAMES and member names from a CMS-like file name using the following scheme:

CMS	TSO Equivalent
fn	PDS member name (ignored if using sequential file)
ft	DDNAME, which in turn points to the TSO dataset
fm	not used on TSO

  - TSO TEST

    You MUST use the “invalid opcode” technique (NOSVC97) when debugging an application which itself uses the TSO TEST facilities. This is because TSO TEST is limited to one use per address space.

- CMS considerations

  IDF/CMS by default uses the invalid opcode technique, which is “full function” on CMS. Instead of using ESTAE/ESPIE on CMS, IDF steals the Program New PSW. IDF/CMS also uses the CP PER/TRACE facilities. This technique is required for debugging read-only code (e.g. within a DCSS) on CMS. (MVS/TSO unfortunately doesn't expose any PER facilities to an application program.)

- VSE considerations

  The program is first link edited with a special version of the VSE/ESA Linkage Editor (ASMLKEDT) that captures external symbols and places that information in the librarian member `phasename.MAP`.

  - VSE naming conventions

    File naming conventions are derived from their CMS equivalents:

CMS	VSE Equivalent
fn	VSE librarian member name
ft	DLBL name, which in turn points to the VSE dataset name
fm	not used on VSE

- ISPF Considerations

Chapter 21 of the IDF User's Guide briefly discusses using ASMIDF with ISPF (for TSO) applications. The invocation command is different, depending on whether the application being debugged resides in the STEPLIB or ISPLLIB allocations. The manual also discusses the use of TSOEXEC, breakpoint method selection, and an example of debugging ISPF dialogs.

ASMIDF does not use ISPF services. It is a TSO Command Processor and will assume control of the entire screen. So if you had a split screen under ISPF and started up ASMIDF on one of the logical screens, the other logical screen(s) would not be available for display until you exited ASMIDF. It may be useful to look at the SWAP option; there is a short section “Programs performing Full-screen I/O” on page 44.

- CICS Considerations

  IDF may be used to debug CICS only if you run IDF on a TSO logon and run then CICS as a program within the TSO region. IDF is not intended for debugging CICS transactions in a production environment.

- DB2 Considerations

  The IDF Reference manual discusses using ASMIDF with DB2 applications (for MVS).

  1. The IDF option NOSVC97 is required.
  2. When testing under LE/370, the LE options NOSPIE and NOSTAE must be used.

- IMS Considerations

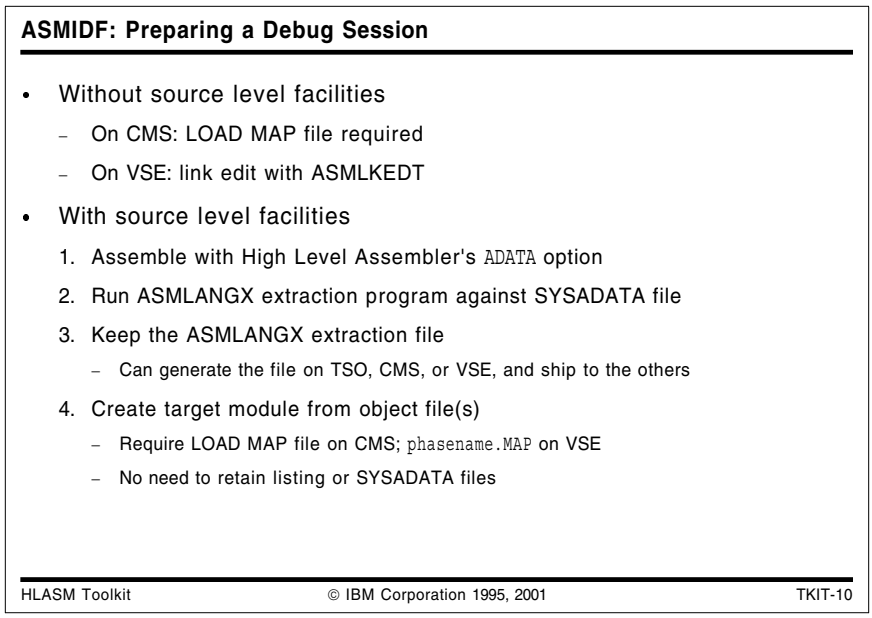
  While IDF has not been tested in the IMS environment, it should be possible to debug Batch Message Programs that run under TSO.

- Authorized code

  IDF as shipped is not authorized and hence will not debug programs that use authorized services.

- Language Environment (LE)

  Just specify the LE option TRAP(OFF) (or the options NOSPIE and NOSTAE), so that Interactive Debug Facility can gain control on breakpoints and other exceptions.



Preparing a Debug Session

ASMIDF may be used to debug a program at the assembler object-code level.

- On CMS the LOAD MAP file must be retained; it is used to determine the location of the program's CSECTs and external symbols. The LOAD MAP file should be renamed so that the file name matches that of the executable module (MODMAP option)
- On TSO, ASMIDF extracts the required information from the load module itself and no extra information is required.
- On VSE, link edit the program with the supplied ASMLKEDT link editor, to capture information about external symbols in the output phase.

To use the source level facilities of ASMIDF, some preparation is required:

1. The assembly must be done with the ADATA option specified and the resultant SYSADATA file used as input to the next step. (The ADATA option and the characteristics of the SYSADATA file are described in the HLASM Programmer's Guide, SC26-4941.)

   Note: There is no special support in IDF for labeled and dependent USING statements.

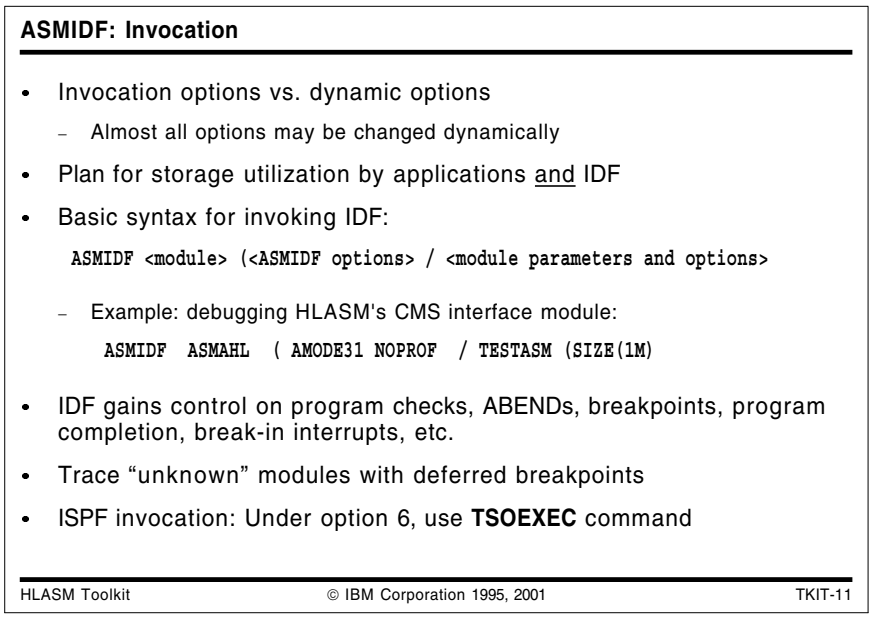
2. Run ASMLANGX using the SYSADATA file as input. This will create an extraction ASMLANGX file that will be used during the debugging session. (The SYSADATA and ASMLANGX files should have the same name.)

   Note: An appendix in the IDF User's Guide describes some useful EXECs.

3. Create the target module from the object-file text as normal; on CMS, retain the (renamed) LOAD MAP file, and on VSE retain the `phasename.MAP` file.

4. The only file required by ASMIDF for source level debugging is the ASMLANGX file; you may erase both the LISTING and SYSADATA files, if desired.

The extraction file produced by ASMLANGX may be created on any MVS, CMS, or VSE system and then be shipped to any of the others.



ASMIDF Invocation

While most option settings may be changed while ASMIDF is running, some may only be set on the command line, for example, AMODE31.

Also note that some programs will consume all available storage, leaving none available for operation of ASMIDF. There are two ways of dealing with this:

1. Load all required files before allowing the program to commence
2. Reduce the storage that the program will obtain.

The debugger always starts in control, and will set up the traps/intercepts that it needs before handing over control to the user program. If the user program then sets up its own traps/intercepts, subsequent actions depend on the underlying operating system.

ASMIDF initializes itself so that if any “interesting event” occurs within the target module, ASMIDF will receive control. Such an event could be any one of the following:

- Program check
- ABEND
- Breakpoint reached (including Watchpoints)
- Program completion
- Break-in interrupt
- Module load (for deferred breakpoints)
- PER interrupt (CMS only)

Unless one of these events occurs, the target program executes without interference from ASMIDF and generally without degradation (slightly dependent on PER options used in CMS).

If you are trying to follow execution through a routine that is “unknown” to IDF, it checks to see that the PSW remains within the program's defined limits and will warn you if you're about to go outside those bounds. The warning is just to let you know that IDF is about to lose control of the session; you can choose to continue if you want. There are several ways around this “unknown routine” problem:

1. Ensure that IDF knows about all modules you'd like to trace. Using DBREAK will help, as IDF sets up the appropriate control blocks itself (TRIGGER LOAD may also help).

2. You can tell IDF about any loaded modules via the SET MODULE command.

```
SET MODULE name BASE address   will tell IDF the start address
SET MODULE name SIZE llllll    will tell IDF the length
```

3. `SET TRACEALL ON` will allow IDF to trace anywhere.

Under ISPF, it is recommended that you invoke ASMIDF via the TSOEXEC command: from option 6 under ISPF, issue:

```
TSOEXEC ASMIDF IEFBR14
```

You may also invoke ASMIDF with the NOSVC97 option: from option 6 under ISPF, issue:

```
ASMIDF IEFBR14 (NOSVC97
```

but this requires certain limitations on the target program's behavior.

ASMIDF: Useful Options

PROFILE/NOPROFIL
IDF by default looks for PROFILE ASM (a REXX exec)

AMODE24/AMODE31
Sets initial AMODE of target program

AUTOSIZE/NOAUTOSZ
Controls automatic window resizing

PATH, FASTPATH
Counts number of instruction executions

LIBE Specifies library containing target application module

CMDLOG, RLOG
Create or append to or replay command log file

HLASM Toolkit TKIT-12

Useful ASMIDF Options

- PROFILE/NOPROFIL

  By default, ASMIDF will run a REXX EXEC named PROFILE ASM during its initialization. This EXEC may be used to customize the environment to your particular needs.

  The PROFILE option allows you to specify a different filename while the NOPROFIL option disables any profile invocation.

  Note: No error messages are issued if the profile is not found.

  Note: No profile is provided with the toolkit; however, a sample profile is illustrated in Figure 9 on page 24.

- AMODE24/AMODE31

  If your target program needs to be started in a particular mode, then use one of these options to set the initial addressing mode.

- AUTOSIZE/NOAUTOSZ

  By default, ASMIDF will AUTOSIZE the displayed windows as windows are opened and closed. You may decide that you'd like to keep the screen layout consistent with

particular windows in specific places; in this case the NOAUTOSZ option stops ASMIDF re-sizing the windows.

- PATH, FASTPATH

  This option provides the user with two new facilities:

  1. ASMIDF will display the number of times that an instruction has been executed.
  2. ASMIDF retains a history of the last 1023 instructions executed. This history may be accessed via the HISTORY command.

  There are some additional variations on PATH that may be useful: PATHFILE and FASTPATH.

- LIBE

  This option will tell ASMIDF to load the target module from the specified library, rather than using the default search order. This is useful on TSO if your test library is not in the default search order.

- CMDLOG and RLOG

  These two options provide a record and playback facility.

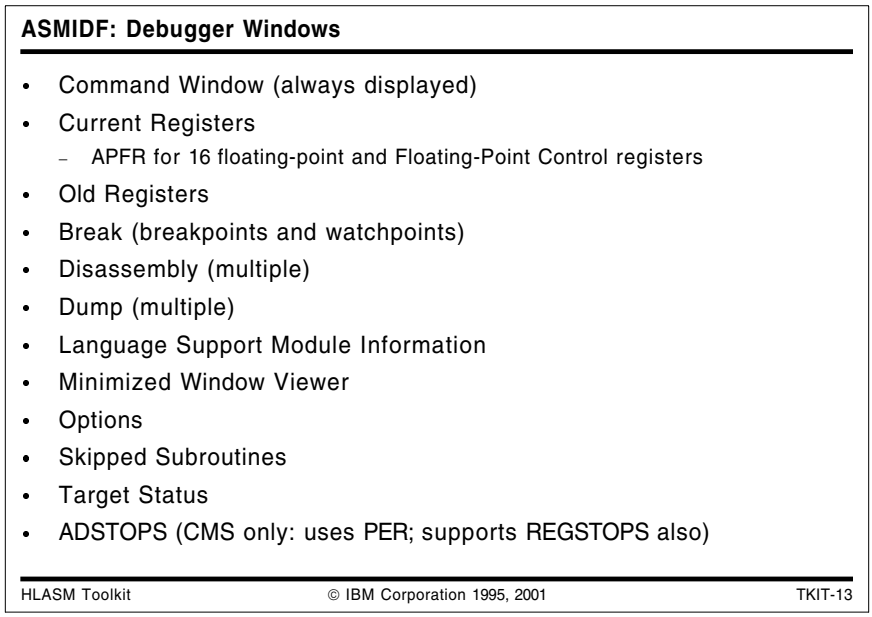
  CMDLOG will cause ASMIDF to log each command in a log file (on CMS, ASM CMDLOG fm; on TSO, the dataset defined by the CMDLOG DD name; and on VSE, the dataset defined by the CMDLOGO DLBL name).

  If RLOG is specified, then once the PROFILE has completed and the target is ready for execution, all the commands in the log file will be replayed.

  Note: CMDLOG will ***append*** to an existing log file. This can cause unexpected results when the log file is then used by RLOG.

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*   This is a sample PROFILE ASM.  To try it, pick your favorite     */
/*   module and then issue:                                           */
/*                                                                    */
/*    ASMIDF module (profile sampprof                                 */
/*                                                                    */
/*--------------------------------------------------------------------*/

 'SET PFK 2 Macro REGS'          /* Define a new PF key - see User  */
                                 /* Guide p241 for REGS macro       */

 'COLOR WRYG'                    /* Customize the colors            */

 'SHOW SOURCE'                   /* Suppress disassembly display    */

 'SET HEXDISP ON'                /* Display all output in hex       */

 'SET HEXINPUT ON'               /* Numeric input default is hex    */

 'SET MSG <<< This is the sample profile >>>'
 Exit
```

Figure 9. Sample profile for ASMIDF



ASMIDF: Debugger Windows

ASMIDF is cursor sensitive: if an argument is missing from a command then it will use the current cursor position and attempt to derive the argument from that.

Some commands allow the user to specify which window the command should apply to. This is done by adding an equal sign followed by the window number. For example, `CLOSE =3` will close window number 3. (The window number is displayed as the first part of the title).

Opening and closing of windows is done by:

1. Issuing the appropriate command for that window. These commands act as toggles which means that if the specified window is not open, then it will be opened; otherwise the specified window will be closed.

   For example, the command REGS will cause the current register window to be displayed (provided that the Current Registers window is not already open).

2. Issuing the OPEN command with the desired window type will open the window if possible (for example, OPEN DUMP).

3. Issuing the CLOSE command against the window.

Most windows will only allow one window of that type to be displayed at a time. However, it is possible to open multiple disassembly and dump windows at once. (The MINimize, MAXimize and ORDER commands may be helpful in this situation to improve readability).

An example of a screen containing multiple windows is shown in Figure 10 on page 26.

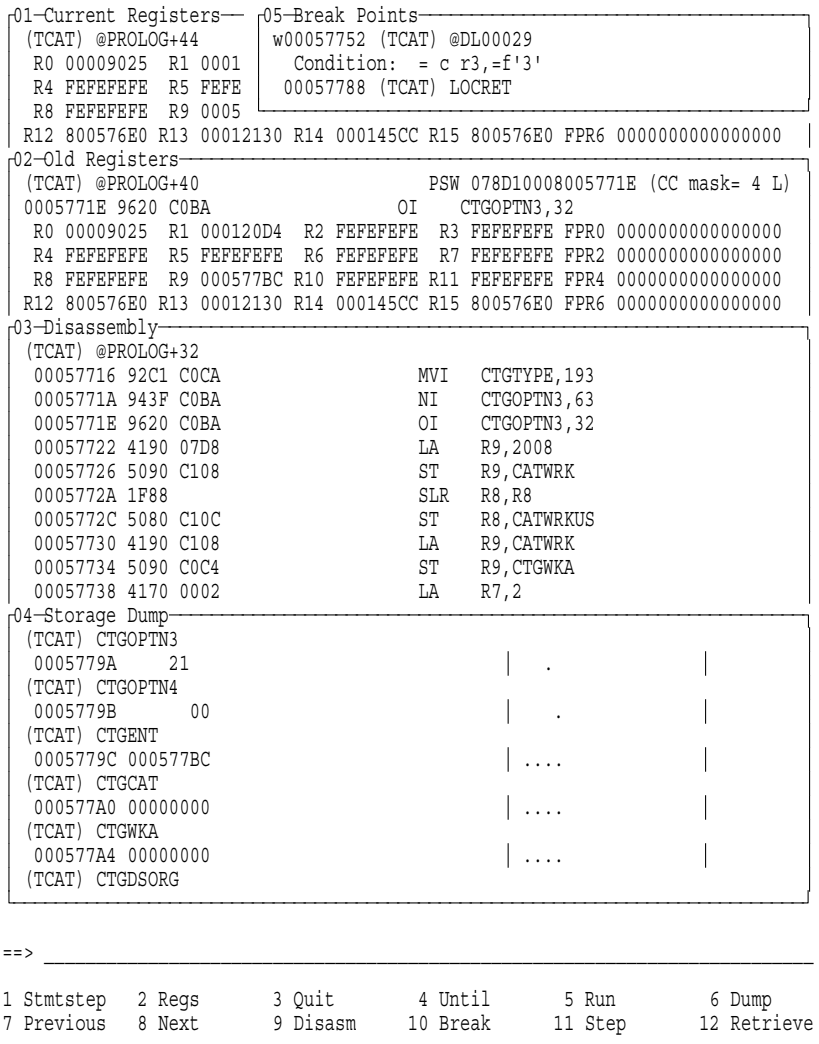


Figure 10. Example of Several Open IDF Windows on One Screen

A brief description of each window type follows. By default, the windows are positioned one after another vertically, except that the AdStops, Break, and Skipped Subroutines windows are positioned at the right edge of the screen.

- Command Window (always displayed)

  The Command Window contains the command input area, the message display area, and the PF-key settings (this portion may be customized).

- Current Registers

  The Current Registers Window displays the current PSW, General Purpose, and Floating Point registers. The Control and Access registers can also be displayed. The contents of the PSW or registers can be modified simply by overtyping.

- Additional Floating-Point Registers (AFPR)

  If AFPR support is available on the processor, all sixteen floating-point registers and the Floating-Point Control Register are displayed and may be updated by overtyping.

- Old Registers

  The Old Registers Window shows the value of the PSW and the General and Floating Point registers the last time IDF was in control. If your program is “single stepping,” the contents of this window are the “before” values prior to executing the current instruction.

- Break (breakpoints and watchpoints)

  The Break Window lists active breakpoints and watchpoints, along with any commands associated with them.

- Disassembly (multiple)

  The Disassembly Windows display storage contents as disassembled Assembler Language instruction statements. Locations at which breakpoints or watchpoints have been set are highlighted. Modifications can be made by overtyping the instruction.

- Dump (multiple)

  The Dump Windows display storage in dump format (both hexadecimal or character). Modifications can be made by overtyping either portion of the display.

  An example of a screen showing storage dumps of two modules is shown in Figure 11.

```
01 Storage Dump
 (ASMXDACP) ASMXDACP
  0015FB18 47F0F0DE D3898385 95A28584 40D481A3 | å00úLicensed Mat |
  0015FB28 85998981 93A24060 40D79996 978599A3 | erials - Propert |
  0015FB38 A8409686 40C9C2D4 40C1E2D4 D3C1D5C7 | y of IBM ASMLANG |
  0015FB48 E7404DC3 5D40C396 97A89989 8788A340 | X (C) Copyright  |
  0015FB58 C9C2D440 F1F9F9F5 4B40C193 9340D989 | IBM 1995. All Ri |
  0015FB68 8788A3A2 40D985A2 8599A585 844B40E4 | ghts Reserved. U |
  0015FB78 E240C796 A5859995 948595A3 40E4A285 | S Government Use |
  0015FB88 99A240D9 85A2A399 8983A385 8440D989 | rs Restricted Ri |
  0015FB98 8788A3A2 406040E4 A2856B40 84A49793 | ghts - Use, dupl |
  0015FBA8 898381A3 89969540 96994084 89A28393 | ication or discl |
  0015FBB8 96A2A499 85409985 A2A39989 83A38584 | osure restricted |
  0015FBC8 4082A840 C7E2C140 C1C4D740 E2838885 |  by GSA ADP Sche |
  0015FBD8 84A49385 40C39695 A3998183 A340A689 | dule Contract wi |
  0015FBE8 A38840C9 C2D440C3 9699974B 400007FE | th IBM Corp. ..Ú |
 (ASMXMAIN) ASMXMAIN
  0015FBF8 47F0F016 10C1E2D4 E7D4C1C9 D54040F9 | å00..ASMXMAIN  9 |
  0015FC08 F54BF2F9 F60090EC D00C18CF 41B0CFFF | 5.296.˚Ö}..õ.{õ. |
  0015FC18 47F0C028 00163FF8 5870C024 58007690 | å0{....8iø{.ì.Î˚ |
  0015FC28 181D1B10 5A00D000 47D0C040 00000002 | ....!.}.å}{ .... |
  0015FC38 50001000 D20F1048 D04818FD 18D150FD | &...K..ç}Ç.Ù.J&. |
  0015FC48 000450D0 F00898F1 F010D203 D0581000 | ..&}0.q10.K.}ì.. |
  0015FC58 5860D058 58806000 5080D100 58A0D050 | ì-}ìiØ-.&ØJ.ìñ}& |
  0015FC68 4120D212 5020A160 D70C2000 20004190 | ..K.&..-P.....˚  |
  0015FC78 D2835090 A164D779 90009000 4130A110 | Kc&˚.ÀP.˚.˚.....  |
```

Figure 11. Example of IDF DUMP Window

- Language Support Module Information (multiple)

  The Language Support Module (LSM) Window can be opened when language-extraction data is available. It can display the values of symbolic variables, as well as the status of the available information.

- Minimized Window Viewer

  The MINIMIZE command can be used to temporarily minimize a window, to make more space available on the screen for other windows. The Minimized Window shows the type and number of the minimized windows.

- Options

  The Options Window displays the current status of IDF options; some of the options can be modified by overtyping their values.

- Skipped Subroutines

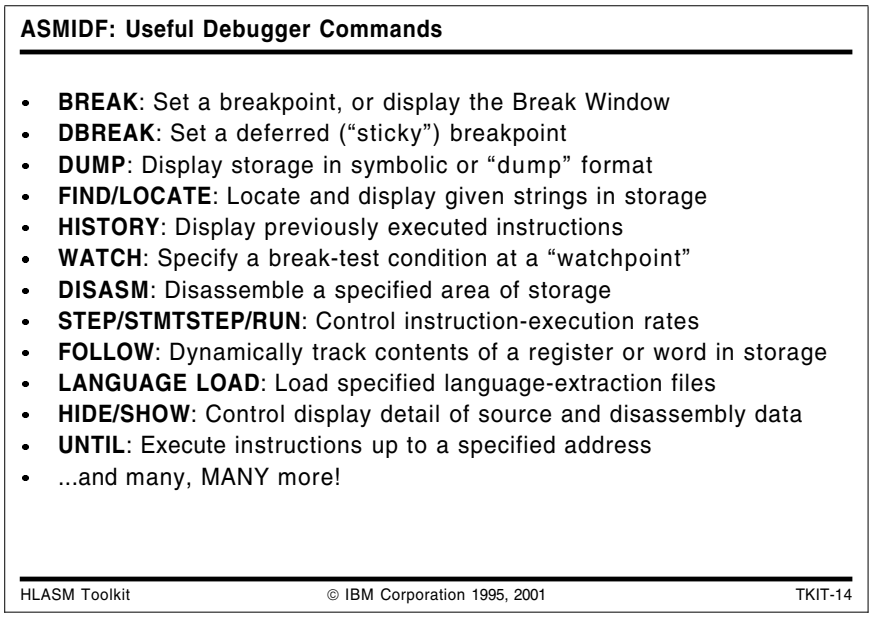
  The Skipped Subroutines Window displays the addresses and names of subroutines for which single-stepping, statement stepping, or instruction counting is being bypassed.

- Target Status

The Target Status Window displays information about all programs known to IDF.

- ADSTOPS (CMS only: uses PER; supports REGSTOPS also)

  The AdStops Window displays the storage ranges to be checked for storage alteration events, and the General Purpose Registers to be checked for register alteration events.



ASMIDF: Useful Debugger Commands

This list shows some of the commands available within ASMIDF. It is by no means comprehensive (there are nearly 190 available commands); the complete list is provided in Chapter 2 of the IDF User's Guide.

You can enter instruction and data addresses symbolically if the Language Support Module (language extraction) is available. This can greatly simplify debugging of “familiar” modules.

Some useful commands are the following:

- BREAK

  Set a breakpoint, or display the Break Window. At most 64 active breakpoints can be set. (In practice, this is many more than normal applications will need.)

- DBREAK

  Set a deferred (“sticky”) breakpoint: these can be used for debugging modules not yet loaded into storage.

- DUMP

  Display storage in symbolic or “dump” format, with overtyping modifications in hex or character format.

- FIND/LOCATE

  Locate and display given strings in storage, using a syntax like that of the ISPF editor FIND command or of the XEDIT LOCATE command.

- HISTORY

Display previously executed instructions when the PATH or PATHFILE option has been specified. This allows you to review the flow of execution that led to the current instruction.

- WATCH

  Specifies a break-test comparison to be checked each time control passes the “watchpoint”; a break occurs only if the condition is true.

- DISASM

  This command requests disassembly of a designated area of storage.

- STEP/STMTSTEP/RUN

  These three commands control instruction-execution rates: RUN executes until the next “event” occurs; STEP executes an instruction at a time; and STMTSTEP executes all instructions associated with a single source-language statement.

- FOLLOW

  The FOLLOW command will cause a Dump Window to automatically track the value of a 4-byte area of storage, or the contents of a register.

- LANGUAGE LOAD

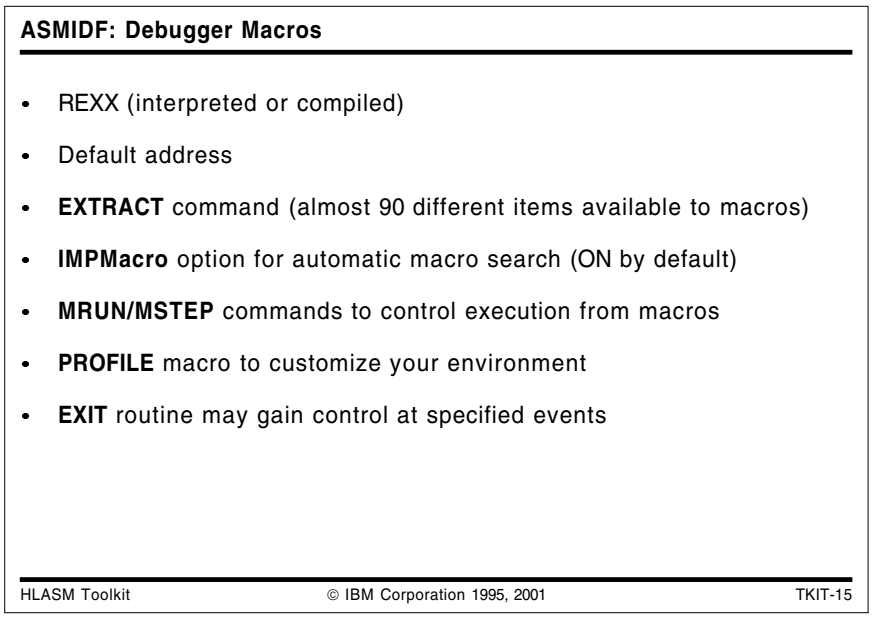
  Loads specified language-extraction files for general or module-specific use.

- HIDE/SHOW

  These two commands control the amount of detail when source code and disassembled storage is being displayed.

- UNTIL

  Executes instructions up to (but not including) a specified address.



ASMIDF: Debugger Macros

ASMIDF provides an extremely flexible and powerful macro facility that you may use to customize your debugging environment. All the macros used by ASMIDF are written in REXX, but you may also write them in “compiled REXX.” Some examples are provided in the IDF User's Guide.

ASMIDF provides many useful facilities to assist the macro writer. Some of these are:

- Default address. ASMIDF sets up a REXX environment that allows the user to direct commands to ASMIDF for processing.

  On CMS, there are some restrictions on the address; these are detailed in Chapter 15 of the User's Guide.
- EXTRACT command. This allows the macro to obtain a great variety of information from ASMIDF about the current environment (see the example below of the REGS macro). Nearly 90 different types of debugger and target-program data are available.
- IMPMacro option. This option (which is set on by default) causes ASMIDF to search for a macro if the entered command is not found in the ASMIDF command table.
- MRUN/MSTEP commands. These cause the target program to immediately resume execution until the next event; control is then returned to the macro.

There are two special macros within ASMIDF; the PROFILE macro and the EXIT macro.

- The PROFILE macro is driven during ASMIDF initialization and may be used to completely customize the user environment.
- EXIT is a special purpose routine which, if enabled via the EXITEXEC command, is given control at various significant events. If the EXIT macro sets a return code of 1, then ASMIDF will NOT display that event to the user and execution of the target will resume as normal. The EXIT routine (whose name is set by the SET EXITEXEC command) may be written in a compiled or assembled language for added convenience or performance, if you specify the CMPEXIT option.

  Chapter 17 of the IDF User's Guide describes EXIT routines.

ASMIDF: Debugger Macros, Example 1

```
/*================================================================\
| TRAP macro:  uses DBREAK to load and break on the entry point of |
|              a loadable module                                   |
| PARAMETERS:  name - module name                                  |
|              symbol - external symbol to set break point on      |
\================================================================*/

arg name symbol .
if name == '' then exit 99
if symbol == '' then symbol = name
'DBREAK ('name'.'symbol')'  /* Issue DBREAK at start of CSECT    */
'MRUN'               /* Program will run until DBREAK is matched */
'QUAL' name                /* Change qualifier                   */
'LAN LOAD' symbol          /* Load extraction file               */
'BREAK' symbol             /* Remove breakpoint at module start */
exit
```


ASMIDF: Debugger Macros, Example 1

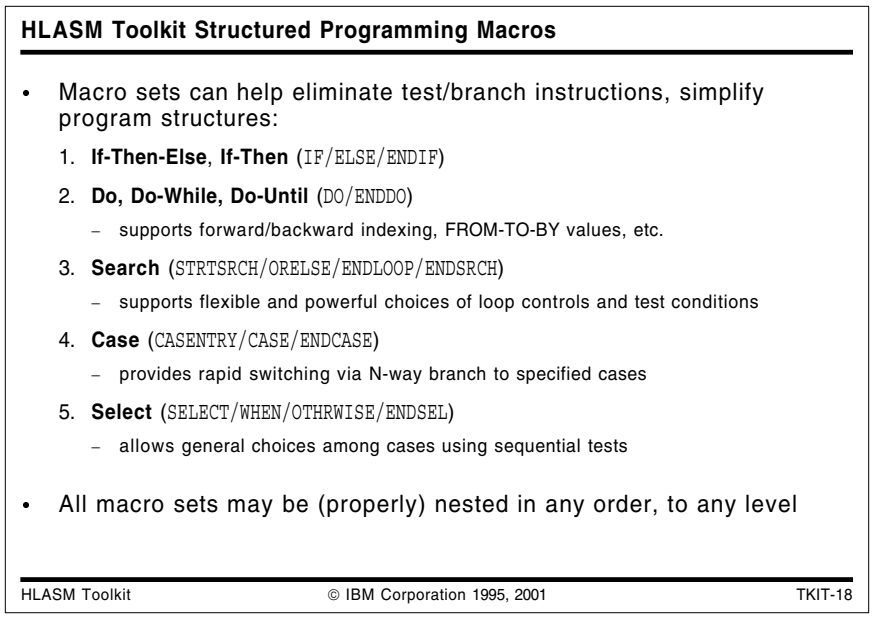
The TRAP macro will set a deferred breakpoint for a module, and then allow the program to RUN until that breakpoint is reached. At that breakpoint, it will change the qualifier for symbols to match the name of the routine to be entered, and then will LANGUAGE LOAD the symbol-extraction file for that section. Finally, it removes the (deferred) breakpoint, and returns control to the user.

ASMIDF: Debugger Macros, Example 2

```
/*REXX ---------------------------------------------------------------*/
/*            REGS - Toggle the current registers window.             */
/*                                                                    */
/* When the REGS window is opened, it will be moved on the ASMIDF  */
/* display so that it is the first window.                            */
/*--------------------------------------------------------------------*/

 'REGS'                          /* Toggle REGS window            */

 'Extract Cursor'                /* Obtain window information     */
 n = Find(display,'REGS')        /* Is REGS window present?       */
 If n ¬= 0 Then                  /* Yes?  Force to be 1st window  */
   'ORDER ='n

 Exit
```


ASMIDF: Debugger Macros, Example 2

The REGS macro (taken from Chapter 16 of the IDF User's Guide) shows how the EXTRACT command may be used to obtain information about the current debugging environment. It checks to see if the REGS window is available, and if so puts it at the top of the display list using the ORDER command.



HLASM Toolkit Structured Programming Macros

The High Level Assembler Toolkit Feature Structured Programming Macros simplify the coding and understanding of complex control flows, and help to minimize the likelihood of introducing errors when coding test and branch instructions.

These macros support the most widely used structured-programming control structures and eliminate the need to code most explicit branches.

The Toolkit Feature Structured Programming Macros can be used to create the following structures:

- IF/ELSE/ENDIF

  One-way or two-way branching, depending on simple or complex test conditions.

- DO/ENDDO and STRTSRCH/ORELSE/ENDLOOP/ENDSRCH

  A rich and flexible set of looping structures with a variety of control and exit facilities.

- CASENTRY/CASE/ENDCASE

  Fast N-way branching, based on an integer value in a register. Deciding which branch to take is made at the CASENTRY macro; a direct branch to the selected CASE is then done, followed by an exit at the ENDCASE macro.

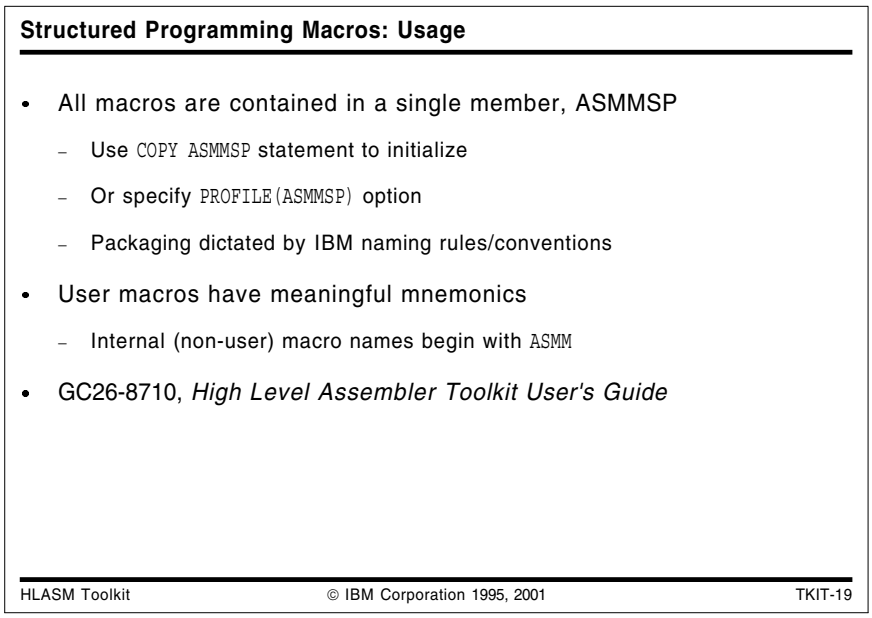
  There is no OTHRWISE facility within this macro set.

- SELECT/WHEN/OTHRWISE/ENDSEL

  Sequential testing, based on sets of comparisons. These macros create a series of tests that are evaluated in the order they are specified in the program. If a test is true, the WHEN section of code for that test will be executed, followed by an exit at the ENDSEL macro.

  If no test is satisfied, then the OTHRWISE section (if present) will be performed.

All the macro sets may be nested, and there are no internal limits to the depth of nesting. Tests made by the various ENDxxx macros ensure that each structure's nesting closure is at the correct level, and diagnostic messages (MNOTEs) are issued if they are not.



Structured Programming Macros: Usage

To use the macros, you must code COPY ASMMSP within the source. This will define all the macros as inline macros. Once this has been done you can use all the macros described without any further limitations. Alternatively, the High Level Assembler PROFILE(ASMMSP) option could be used to automatically include the ASMMSP member into the source without requiring any source changes.

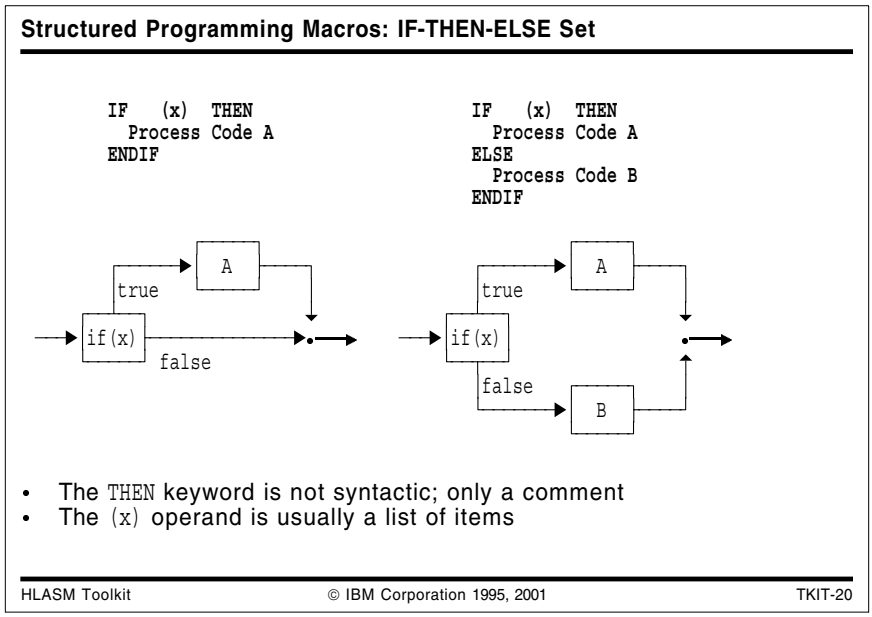
Due to IBM Corporate product-naming standards, all distributed part names must start with the product prefix. In the case of these macros, this resulted in the creation of the ASMMSP member which contains all the “high level” user macros such as IF, CASE, etc. All supplied members have a prefix of ASMM.

The “user” macros are grouped in the following five sets:

- IF/ELSE/ENDIF
- DO/DOEXIT/ENDDO
- CASE/CASENTRY/ENDCASE
- STRTSRCH/EXITIF/ORELSE/ENDLOOP/ENDSRCH
- SELECT/WHEN/OTHRWISE/ENDSEL

We will describe each of these sets in turn.

Note: The structured-programming macros generate base-displacement branch instructions, and therefore assume a base register provides addressability. If you intend to use them in a context where relative branch instructions are desired, you may need to provide local addressability for the macros.



Structured Programming Macros: If-Then-Else

These “IF-THEN-ELSE” macros (IF/ELSE/ENDIF) provide for a one- or two-way branch depending on a condition. You may select execution of one of two blocks of code depending on a true-false condition.

The one-way branch is illustrated in Figure 12:

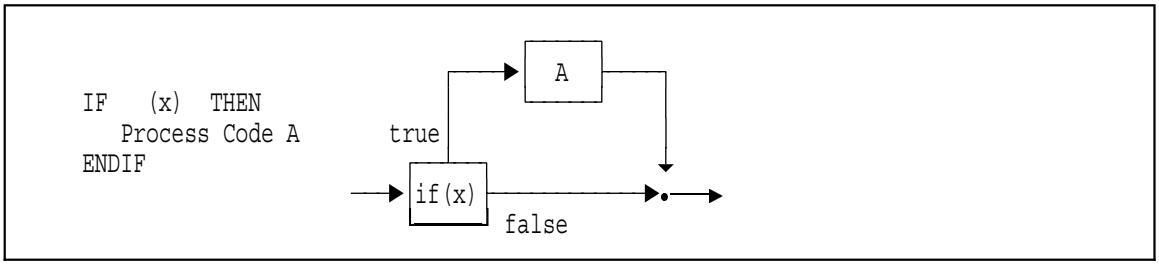


Figure 12. IF-THEN Control Structure

The two-way branch is illustrated in Figure 13:

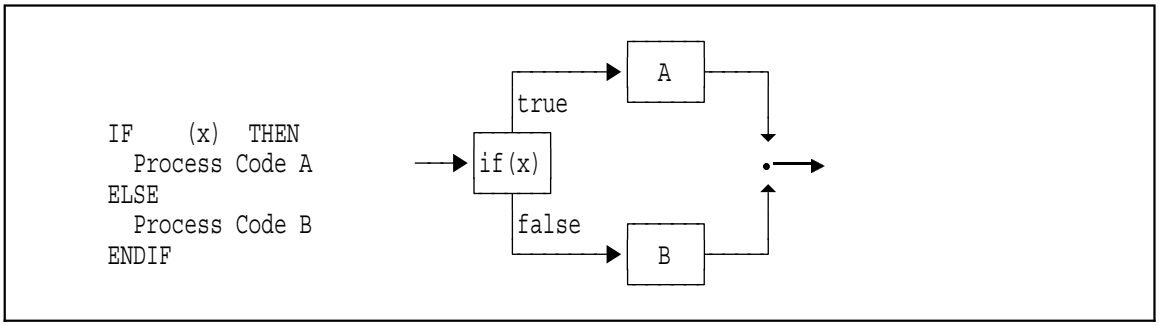
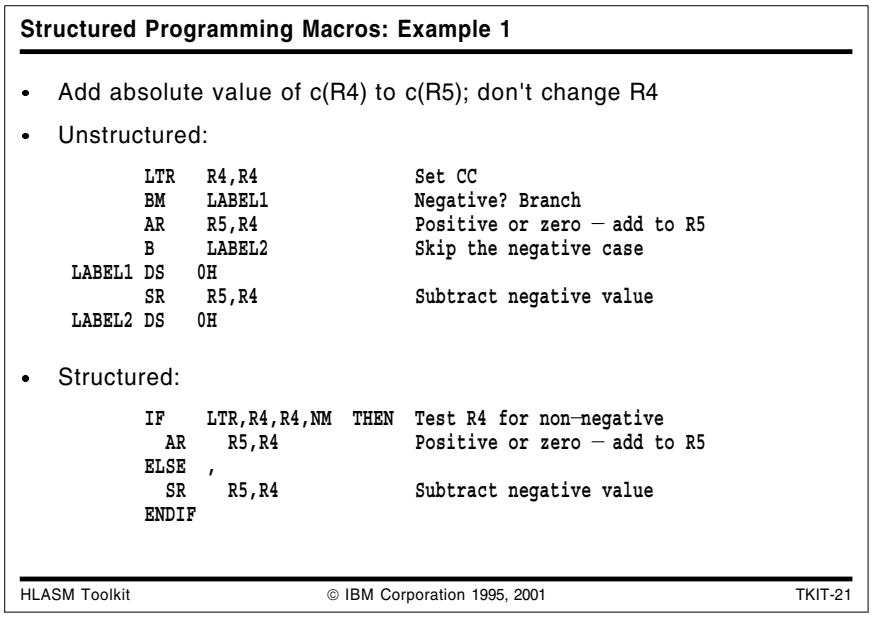


Figure 13. IF-THEN-ELSE Control Structure



Structured Programming Macros: Example 1

This assembler program segment shows how to test a variable and then execute one of two paths depending on the value of the variable. The "problem" requires that we add the absolute value of the contents of R4 to R5, without disturbing R4.

This IF/ELSE/ENDIF structure is first coded using basic assembler language and then using the toolkit macros. The unstructured assembler language segment could appear as follows:

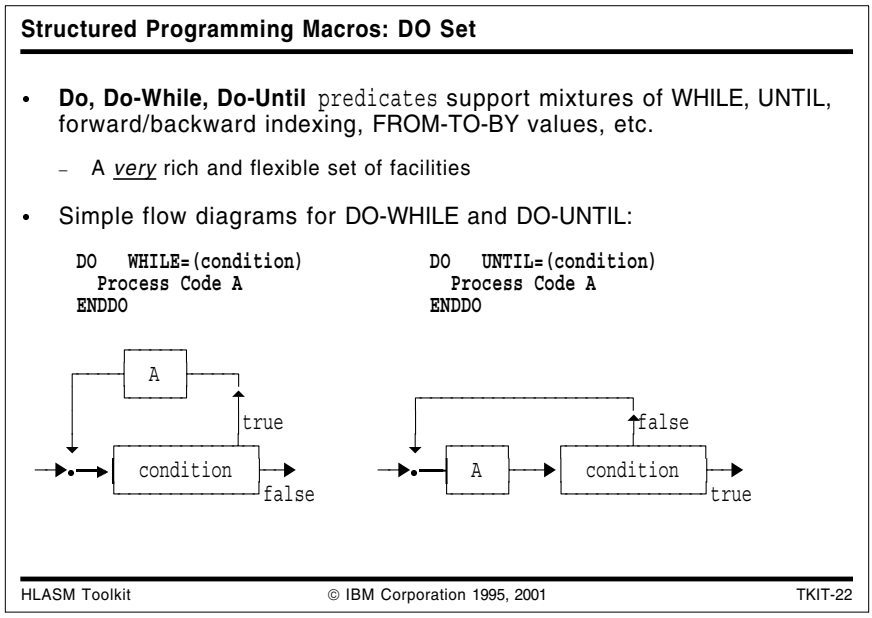
```
        LTR   R4,R4              Set CC
        BM    LABEL1             Negative? Branch
        AR    R5,R4              Positive or zero - add to R5
        B     LABEL2             Skip the negative case
LABEL1  DS    0H
        SR    R5,R4              Subtract negative value
LABEL2  DS    0H
```

The structured equivalent could be written as follows (remember that the THEN "keyword" is only a comment; it is not part of the syntax of the IF/ELSE macros):

```
        IF    LTR,R4,R4,NM  THEN  Test R4 for non-negative
          AR    R5,R4             Positive or zero - add to R5
        ELSE  ,
          SR    R5,R4             Subtract negative value
        ENDIF
```

and the results would be identical to the original (non-structured) statements:

```
        IF    LTR,R4,R4,NM  THEN  Test R4 for non-negative
+            LTR  R4,R4
+            BC 15-11,#@LB1
          AR    R5,R4             Positive or zero - add to R5
        ELSE  ,
+            BC   15,#@LB3
+#@LB1       EQU  *
          SR    R5,R4             Subtract negative value
        ENDIF
+#@LB3      EQU *
```



Structured Programming Macros: Do, Do-While, Do-Until

These macros provide for executing a block of code repeatedly until some limit is reached or some condition is satisfied (DO, DO-WHILE, DO-UNTIL macros). The conditions controlling the looping and the termination condition may be specified in a rich set of combinations:

- with FROM,TO,BY specifications, or with infinite looping
- by counting
- with forward or backward indexing
- with explicit specification of BXH or BXLE
- DO-WHILE and DO-UNTIL (or mixed with any other DO type)

We will illustrate only one of the possibilities here. The Do-While and Do-Until control structures are illustrated in Figure 14 below:

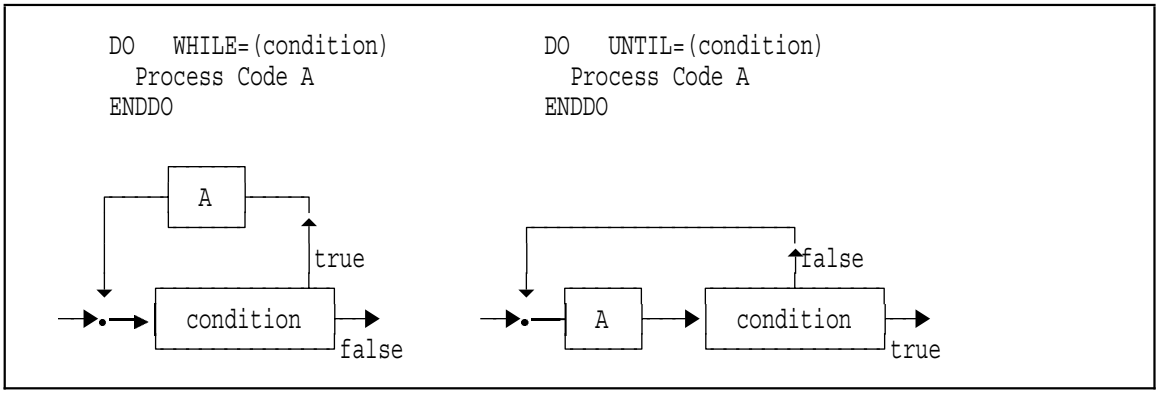
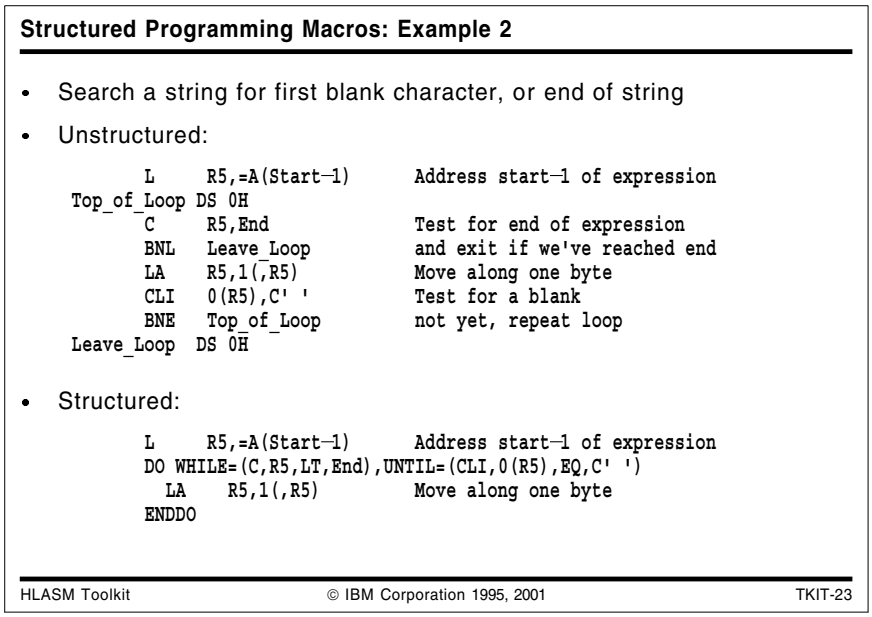


Figure 14. DO-WHILE and DO-UNTIL Control Structures



Structured Programming Macros: Example 2

This assembler program segment shows a simple loop that scans storage until either a blank is found or the end-of-string address is reached.

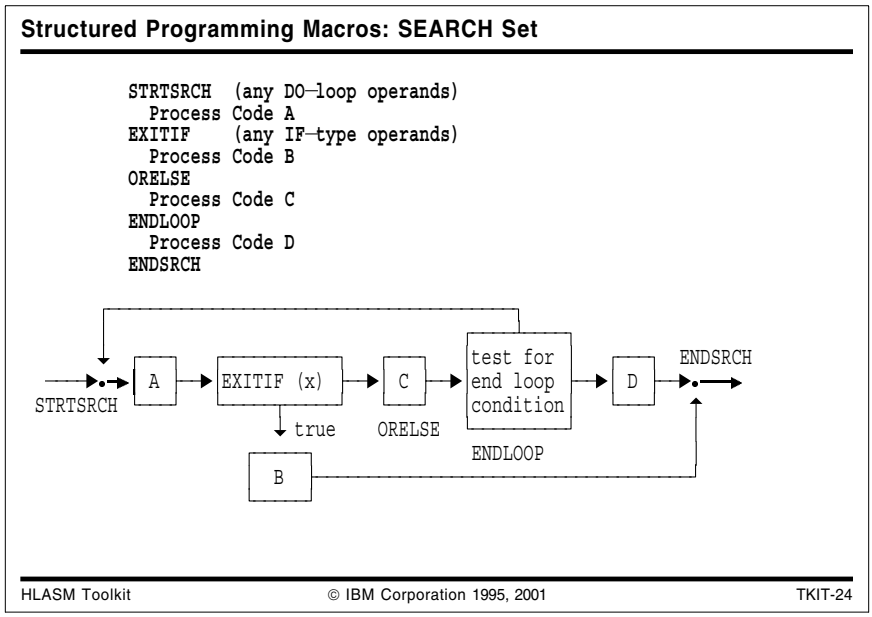
This DO/ENDDO structure is first coded using basic assembler language and then using the toolkit macros. The unstructured assembler language might appear as follows:

```
        L     R5,=A(Start-1)      Address start-1 of expression
Top_of_Loop DS 0H
        C     R5,End              Test for end of expression
        BNL   Leave_Loop          and exit if we've reached end
        LA    R5,1(,R5)           Move along one byte
        CLI   0(R5),C' '          Test for a blank
        BNE   Top_of_Loop         not yet, repeat loop
Leave_Loop  DS 0H
```

The same example could be coded using the DO and ENDDO macros as follows:

```
        L     R5,=A(Start-1)      Address start-1 of expression
        DO WHILE=(C,R5,LT,End),UNTIL=(CLI,0(R5),EQ,C' ')
          LA    R5,1(,R5)         Move along one byte
        ENDDO
```

Note that in both examples the required COPY ASMMSP statement is not shown.



Structured Programming Macros: Search Set

These macros provide for executing a search loop with flexible controls over exit and iteration conditions (SEARCH macros).

The control structure supported by the Search Set is shown in Figure 15 below:

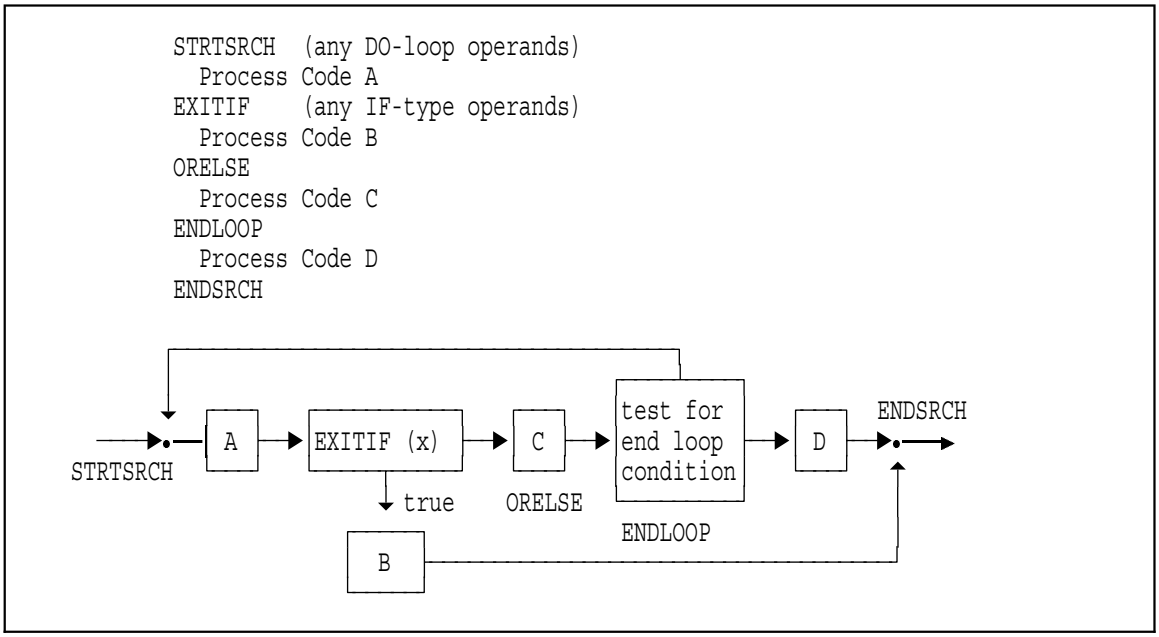
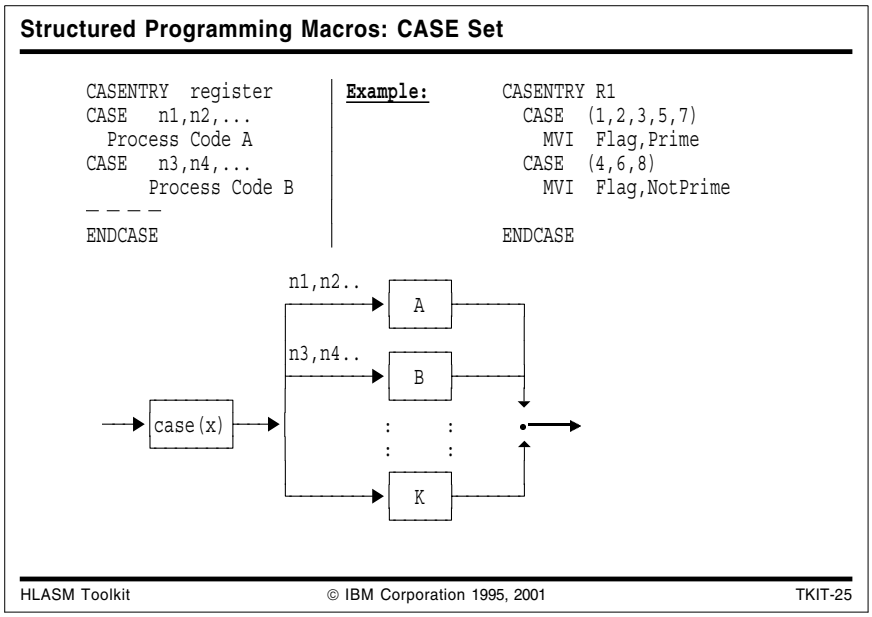


Figure 15. SEARCH Control Structures



Structured Programming Macros: Case Set

These macros provide for executing a block of code selected from a set, based on an integer value contained in a general register. The integer value may also be a power of two, as specified by the optional POWER= keyword.

The selected case is branched to directly, using one of two selection mechanisms depending on whether the branching should use a “vector” of addressable branch instructions (VECTOR=B) or a table of addresses (VECTOR=BR).

The CASE control structure is illustrated in Figure 16 below:

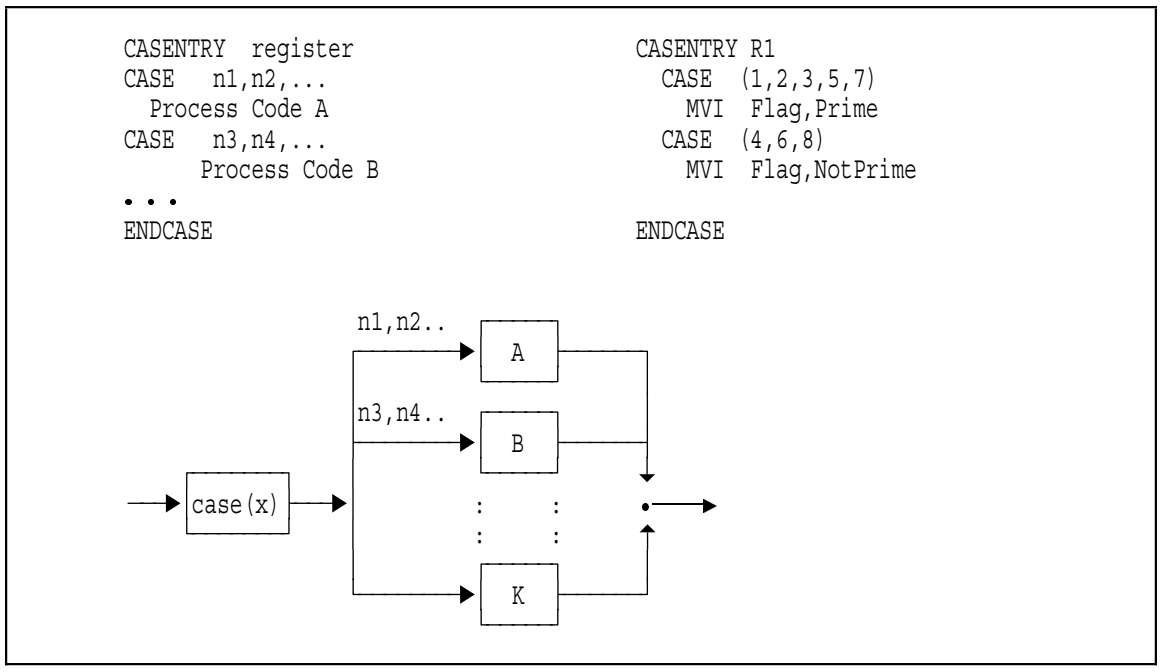


Figure 16. CASE Control Structures

A simple example of the CASE macros is the following:

```
          CasEntry R1
            Case  (1,2,3,5,7)
               MVI  Flag,Prime
            Case  (4,6,8)
               MVI  Flag,NotPrime
          EndCase
          - - -
Flag      DC    X'0'
Prime     Equ   X'80'
NotPrime  Equ   X'40'
```

Figure 17. CASE Example

Structured Programming Macros: SELECT Set

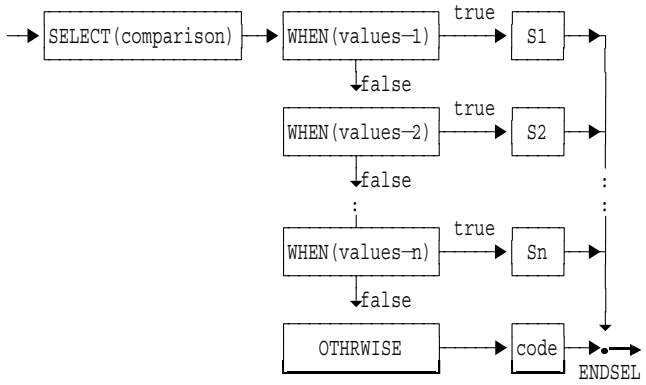
```
SELECT (comparison)          Compare instruction & condition
WHEN   (list-of-values-1)    Values for this comparison
   <statements-1>            Statements for these cases
WHEN   (list-of-values-2)    Values for this comparison
   <statements-2>            Statements for these cases
• • •
WHEN   (list-of-values-n)    Values for last comparison
   <statements-n>            Statements for these cases
OTHRWISE
   <statements>              Executed if no matching WHEN
ENDSEL
```

Example:

```
SELECT  C,R1,Eq
  WHEN  (=F'1',=F'2',=F'3',=F'5',=F'7')
     MVI  Flag,Prime
  WHEN  (=F'4',=F'6',=F'8')
     MVI  Flag,NotPrime
  OTHRWISE
     MVI  Flag,Unknown
ENDSEL
```

HLASM Toolkit © IBM Corporation 1995, 2001 TKIT-26

Structured Programming Macros: SELECT Set ...



HLASM Toolkit © IBM Corporation 1995, 2001 TKIT-27

Structured Programming Macros: Select Set

These macros provide for executing a block of code selected from a set of blocks, based on a varied choice of comparisons.

While they are structurally similar to the CASE set, their behavior is quite different. Each WHEN clause is tested in the order specified until a “true” condition is found, when the corresponding block of statements will be executed; the optional OTHRWISE block is executed if no WHEN clause is true. The structure of the Select Set is illustrated below:

```
SELECT (comparison)           Compare instruction & condition
WHEN   (list-of-values-1)     Values for this comparison
   <statements-1>             Statements for these cases
WHEN   (list-of-values-2)     Values for this comparison
   <statements-2>             Statements for these cases
• • •
WHEN   (list-of-values-n)     Values for last comparison
   <statements-n>             Statements for these cases
OTHRWISE
   <statements>               Statements executed if no matching WHEN
ENDSEL
```

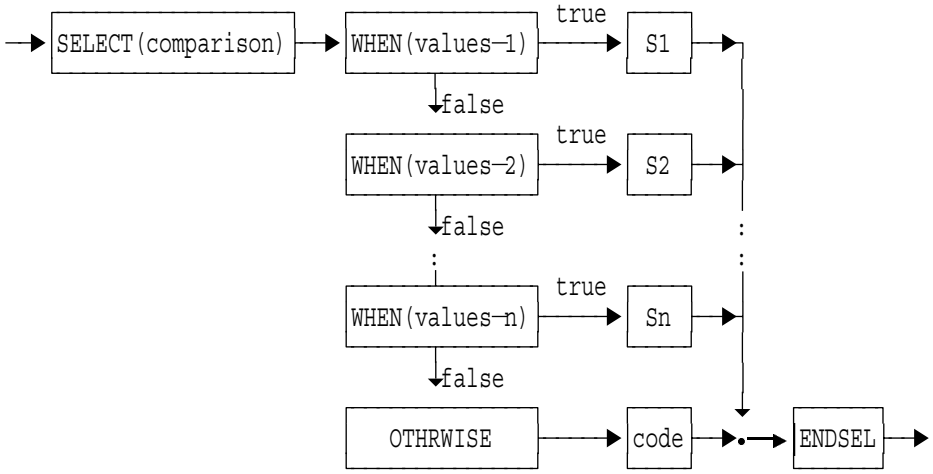
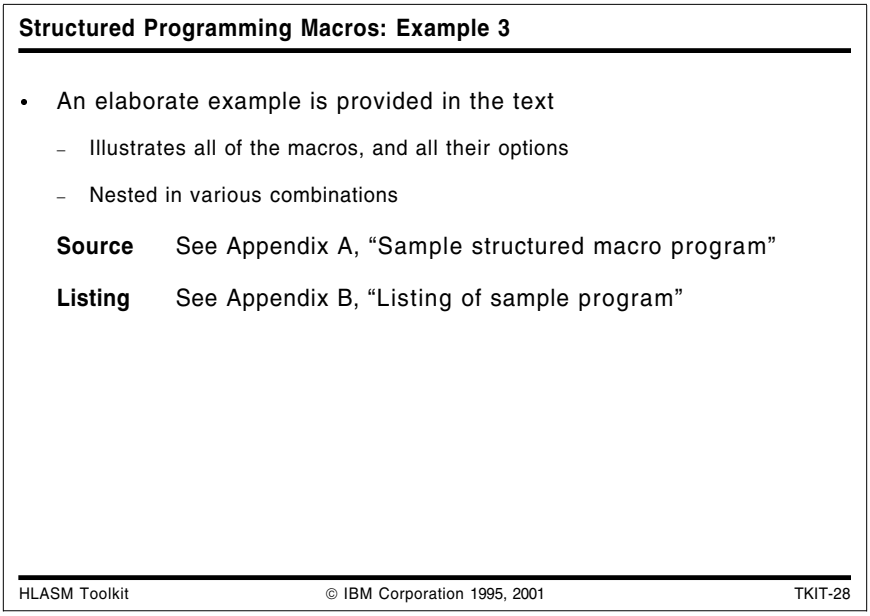


Figure 18. SELECT Control Structures

A simple example of the SELECT macros is the following:

```
          Select  C,R1,Eq
            When  (=F'1',=F'2',=F'3',=F'5',=F'7')
               MVI  Flag,Prime
            When  (=F'4',=F'6',=F'8')
               MVI  Flag,NotPrime
            Othrwise
               MVI  Flag,Unknown
          EndSel
          - - -
Flag      DC    X'0'
Prime     Equ   X'80'
NotPrime  Equ   X'40'
Unknown   Equ   X'01'
```

Figure 19. SELECT Example



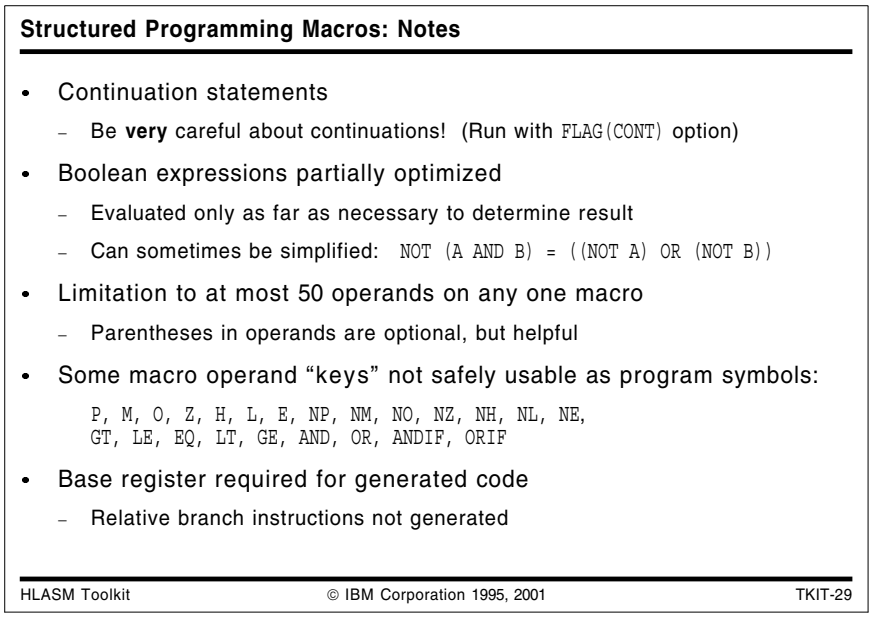
Structured Programming Macros: Example 3

An extensive sample program is provided in Appendix A, “Sample structured macro program” on page 55. It shows the use of more complicated structures and the nesting of macros. (Note that no macros from the SELECT set are illustrated.)

The assembly listing is provided in Appendix B, “Listing of sample program” on page 58. The listing was created by the following (CMS) commands:

1. Access the High Level Assembler Toolkit disk
2. GLOBAL MACLIB ASMSMAC
3. ASMAHL SAMPLE (PROFILE(ASMMSP),NOESD,NORLD,NOXREF,NOMXREF,NOUSING

The expansion of the macros is shown in the listing; if this expansion is not desired then you may use the PC(NOGEN) option to suppress the generated lines.



Structured Programming Macros: Notes

Some minor points are worth remembering:

- Be very careful to place any continued operands in the correct column. The normal assembler rules apply (along with any changes that the ICTL statement may have introduced). The assembly-time option FLAG(CONT) can help determine where the rules have not been followed.
- Not only are the instructions generated by the macros nearly optimal, the macros do not need to evaluate all the terms in a Boolean expression before branching. In the following statement:

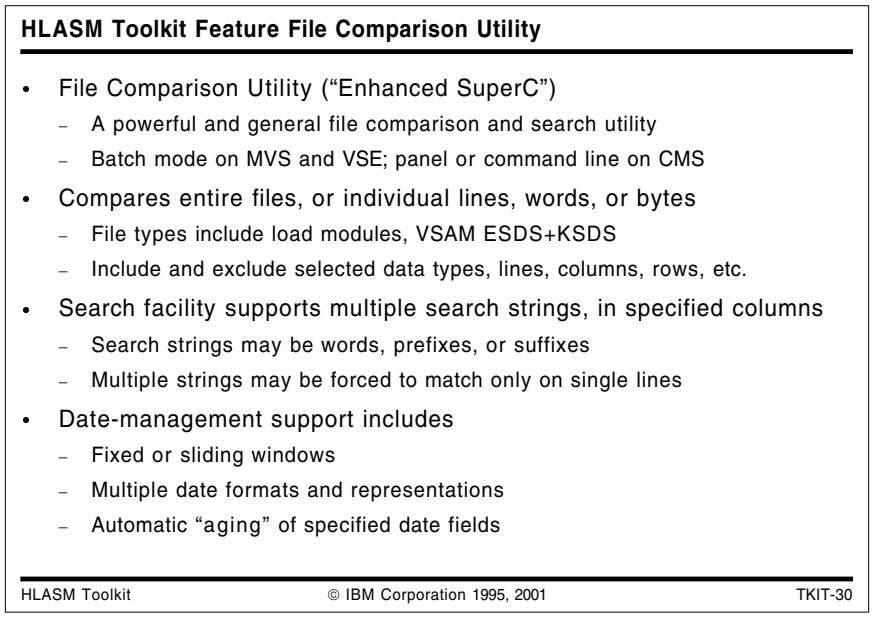
  ```
  IF (LTR,R5,R5,P),AND,(LTR,R6,R7,P)
  ```

  the second load and test (LTR) instruction will *not* be executed if the first LTR sets a negative or zero condition code, as the macros "know" that the expression must return false after only the first part has been evaluated.

  A small reminder about Boolean logic: you can sometime simplify the operands of a test by rewriting expressions:

  ```
  NOT (A AND B)   is equivalent to   ((NOT A) OR (NOT B))
  ```

- Most of the original limitations of these macros have been removed. (They were caused by previous assemblers having fixed array sizes; in HLASM, arrays are dynamic in nature and will grow as required.) One limitation remains: Boolean expressions are limited to fifty (50) operands. This count includes any operators such as AND, OR, etc.
- The use of parentheses in Boolean expressions is optional, but may assist with the understanding of the logic.
- Some "keys" required for correct operand parsing should not be used as ordinary program symbols: `P, M, O, Z, H, L, E, NP, NM, NO, NZ, NH, NL, NE, GT, LE, EQ, LT, GE, AND, OR, ANDIF, ORIF`.
- Because the macros do not generate relative branch instructions, a base register is required for the generated code.



HLASM Toolkit Feature File Comparison Utility

The High Level Assembler Toolkit Feature File Comparison Utility, also known as Enhanced SuperC, is a versatile program that can be used to compare two sets of data (using the Comparison Facility) or to search a specific set of data for a nominated search string (using the Search Facility).

Enhanced SuperC executes in batch mode on MVS and VSE, and on VM via a CMS panel or command line interface. You can compare sequential files, or select multiple or all members of libraries. You can also compare VSAM files on MVS and VSE.

Enhanced SuperC's Comparison Facility requires only the names of the two items to be compared. The Search Facility requires only the name of the item to be searched and the search string to be used. You can tailor the comparison or search according to your requirements, using process statements and process options.

With the Comparison Facility, you can:

- specify the “level” of comparison (file, line, word or byte)
- exclude certain data from the comparison, such as specific sets of rows or columns, or records (such as page headings) containing specified character strings.
- restrict the comparison to certain types of data
- control the type of listing output produced
- specify that an update file be produced
- compare two files that have been reformatted (reformatted files contain such differences as indentation level changes, spaces inserted or deleted)
- detect word changes within documents
- stop immediately when a difference is detected.

Enhanced SuperC's Search Facility lets you specify:

- one or more search strings
- if multiple search strings are sought, whether they are independent of each other or if they must be present on the same line of the member/file being searched
- if the search string is a word, a prefix, or a suffix
- the position in the line of a search string
- the number of lines to be listed which appear before and after the line where the search string was found.

Enhanced SuperC is also a valuable tool for managing year post-Y2K date comparisons (such as for "windowing" of two-digit years). It supports:

- various date formats (particularly in regard to 2-digit and 4-digit year representations)
- a fixed "window" where date comparisons will take place within fixed boundaries
- a sliding "window" where the year range is based on the current year
- the ability to successfully compare files where one file has 4-digit year values and the other has 2-digit year values
- the ability to successfully compare year data where year value is compressed in one file and uncompressed in the other
- the ability to successfully compare data, reports, forms, screens, and panels where data has moved within a line due to the addition of the century digits to 2-digit years.

Complex date comparisons may be performed on dates in many formats, including or excluding specified sets of rows (lines) and/or columns. (The description in the manual section titled "Year 2000 Date Definitions" is very generally useful, despite its title!)

The year Date Aging option "ages" all of the defined dates in either the new or old file: a specified number of years is added to the "year" portion of each designated date in the file before they are compared.[1]

Date Definition statements define the location and format of date fields in the input file. Dates may be described in a wide variety of formats, including allowing "separator" characters (such as : or /). Internal representations supported include character, zoned decimal, packed decimal, and unsigned packed decimal (hex).

The Y2PAST option uses a fixed or sliding window to specify a 100-year period for determining the century-part of a date when only a 2-digit year appears in the data.

---

[1] Aging dates and comparing them is not always straightforward, due to leap years. See Appendix A ("Other Matters to Consider Before You Test") in the Redbook *VisualAge 2000 Test Solution: Testing Your Year 2000 Conversion*, SG24-2230-01.

HLASM Toolkit Feature Usage Scenarios

1. ***Recovery*** from object/load modules (if original source is lost)
   - **Disassembler** initially produces "raw" Assembler source from "binary"
   - Control statements define code, data, USINGs, labels, DSECTs, etc.
   - Repeat disassembly/analysis/description/assembly cycle until satisfied
2. ***Analysis and understanding*** of Assembler Language source programs
   a. **ASMXREF** cross-reference token scanner
      - Locates important symbols, user-selected "tokens"
      - Creates "impact-analysis" spreadsheet-input file for effort estimation

   b. **ASMPUT** Program Understanding tool
      - Graphic displays of program structure, control flow, with any level of detail
      - Can be used to ***help*** reconstruct (lost) source in HLLs!
3. ***Modification, testing, and validation*** of updated programs
   - **Interactive Debug Facility** speeds and simplifies program testing
   - **Structured Programming Macros** clarify program coding logic
   - **File Comparison Utility** tracks before/after status of source, outputs

HLASM Toolkit TKIT-31

HLASM Toolkit Feature Usage Scenarios

We will describe how you might use the High Level Assembler Toolkit Feature for typical program recovery, development, analysis, conversion, and maintenance tasks. These three scenarios show how the challenges of such tasks can be completed with greater speed and simplicity using the Toolkit Feature.

The Toolkit Feature components will be described in three scenarios:

- recovery and reconstruction of symbolic Assembler Language source code
- analysis and understanding of complex Assembler Language programs
- modification, testing, and validation of applications.

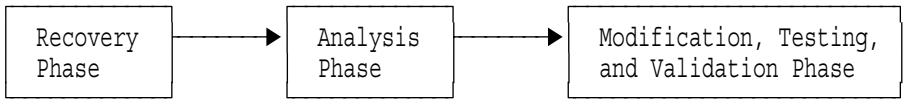


Figure 20. Typical Scenarios for Toolkit Feature Usage

1. ***Recovery and reconstruction*** of Assembler Language source statements from object/load modules for which the original source is lost. The Disassembler initially produces non-symbolic Assembler Language source from object code. You can add control statements iteratively to help define code, data, USINGs, labels, and DSECTs symbolically.
2. ***Analysis and understanding*** of Assembler Language source programs can benefit from three Toolkit components: the Cross-Reference Facility, the Program Understanding Tool, and the Interactive Debug Facility.

   a. The Cross-Reference Facility source analyzer and token scanner can be used to locate important symbols, user-selected tokens, macro calls, inter-module references, and other helpful data. ASMXREF can also create an "impact-analysis" file for input to a spreadsheet application for effort estimation and impact assessment. Another ASMXREF output is a tagged Assembler Language source file: when assembled with the ADATA option, this file produces a SYSADATA file for you to use with the Program Understanding Tool.

b. The Program Understanding Tool provides graphic displays of program structure, control flow, a simplified listing, and other views with any desired level of detail. With the ADATA file created from the tagged source produced by ASMXREF, key areas of the program can be rapidly located and analyzed.

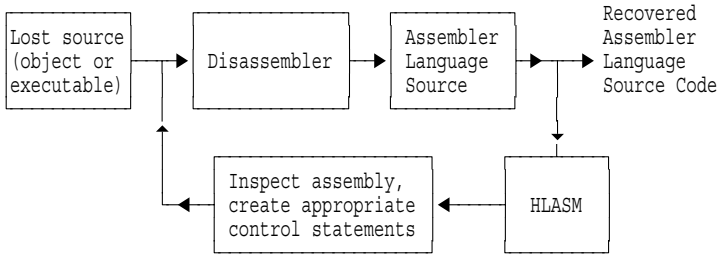
c. The Interactive Debug Facility is by design a “program understanding” tool that lets you monitor the behavior of programs at every level of detail. Data flows may be monitored and traced among registers and storage, even showing the operations of individual instructions!

Note that the combination of Disassembler, Cross-Reference Facility, and Program Understanding Tool can be used to help reconstruct lost source in compiled High Level Languages.

3. ***Modification and testing*** of updated programs is simplified by using the powerful Interactive Debug Facility. At the same time, program logic can be simplified by replacing complex test/branch logic with the Structured Programming Macros.

   Validation: At each stage where the application has been changed, you will probably want to compare its “pre-modification” output to its “post-modification” output, retaining the output files (sometimes called “base logs”) for subsequent validation tests. The File Comparison Utility Enhanced SuperC is designed specifically for such tasks.

HLASM Toolkit Feature: Recovery and Reconstruction



- Start with object code (object files or executables)
- Disassemble and inspect; create control statements to describe the program more fully
- Repeat this cycle as more of the program is understood
- Readable source is used as input to later phases

HLASM Toolkit TKIT-32

Recovery and Reconstruction

During the Recovery and Reconstruction phase, you will typically begin with a program in object or executable format (except CMS MODULEs). Using the Disassembler and by providing suitable control statements, you can create an Assembler Language source program with as much structure and symbolic labeling as you like.

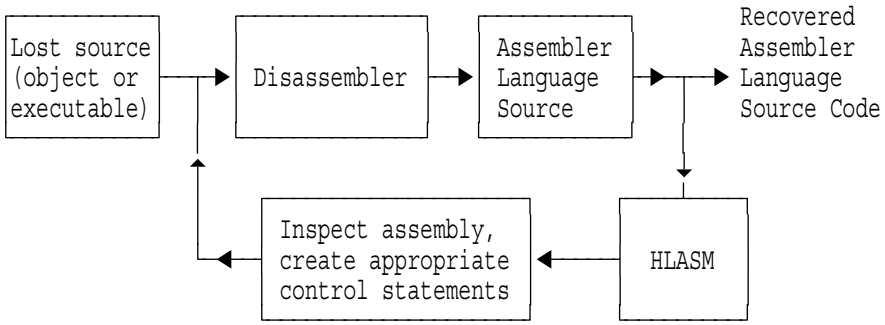
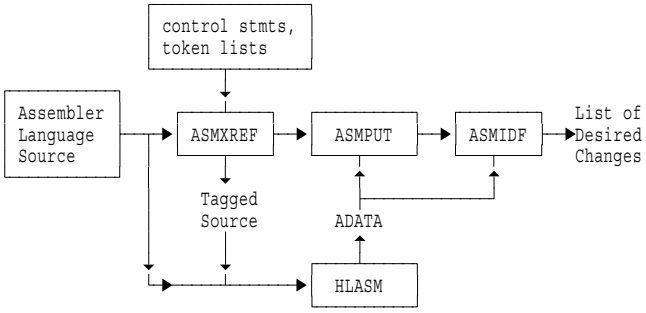


Figure 21. Toolkit Feature: Recovery and Reconstruction Scenario

The disassembly/analysis/description/assembly cycle may be repeated until satisfactory Assembler Language source code is obtained.

The initial steps do not require reassembly of the generated Assembler Language source, as appropriate control statements are usually easy to determine from the Disassembler's listing. As the recovered program approaches its final form, you should assemble it with HLASM to ensure the validity of your new source program.

HLASM Toolkit Feature: Analysis and Understanding



- ASMXREF scans assembler source programs, identifies key items
  - Create “tagged” source file identifying important “tokens”
- Assemble; ASMPUT uses ADATA to analyze control flows
- Use IDF to trace data flows in detail

HLASM Toolkit TKIT-33

Analysis and Understanding

The most complex aspect of application maintenance and migration is analyzing and understanding the code. There are three components of Toolkit Feature that can help:

- ASMXREF can locate all uses of a variable name or any character string. A tagged Assembler Language source program may also be produced.
- ASMPUT provides graphical views of control flows within and among programs and modules.
- The Interactive Debug Facility helps you monitor and track the behavior of individual instructions and data items.

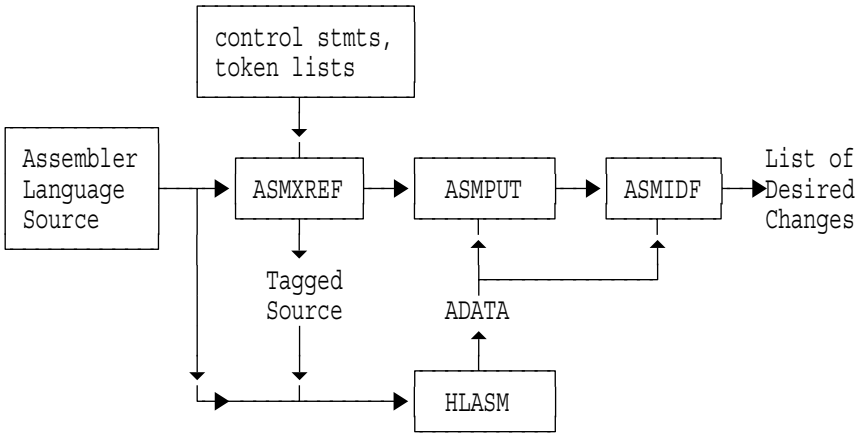
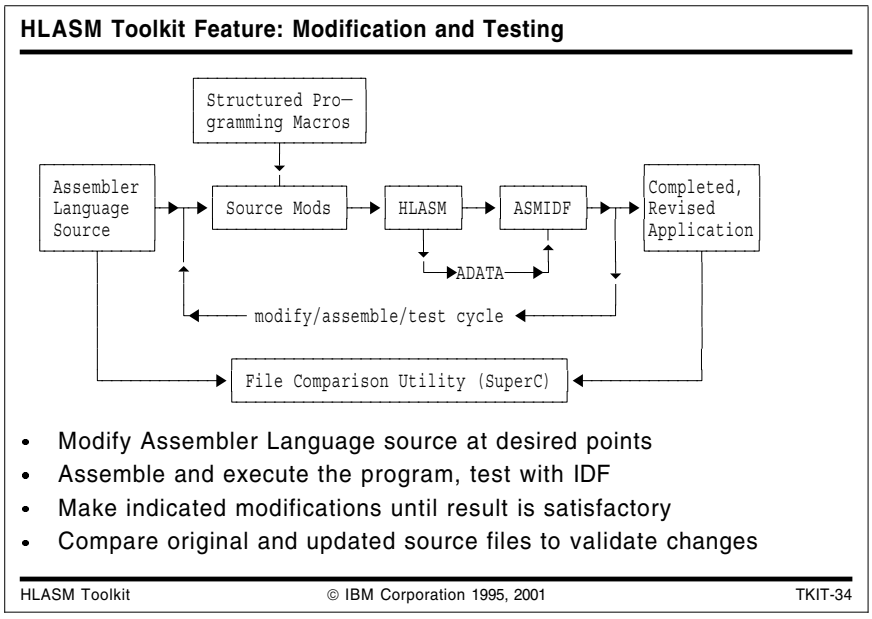


Figure 22. Toolkit Feature: Analysis and Understanding Scenario

While each of these components has valuable capabilities, using them in combination can provide great synergy in analyzing and understanding program behavior.



Modification and Testing

After you have used the Toolkit's disassembler, ASMXREF, and ASMPUT components to determine the needed modifications, the Structured Programming Macros can be added to simplify the coding and logic of the program.

The Enhanced SuperC comparison utility can then be used to compare the original and updated source files to validate the placement and coverage extent of all modifications.

You can then test the updated code using the rich and flexible features of the Interactive Debug Facility. After each assembly/debug cycle, you can further modify the source code, repeating the process until the completed application is accepted for installation in a production library.

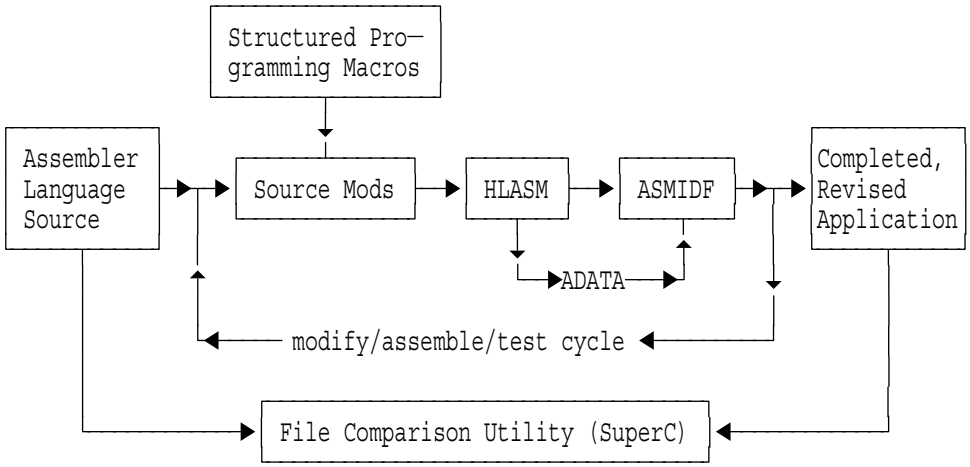
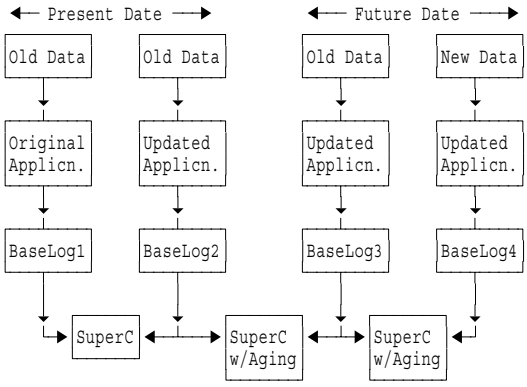


Figure 23. Toolkit Feature: Modification and Testing Scenario

HLASM Toolkit Feature: Validation



- Create “base logs” with original and updated application, current and “future” dates, and old and modified data
- Compare results at each stage using “Aging” facilities as needed

HLASM Toolkit TKIT-35

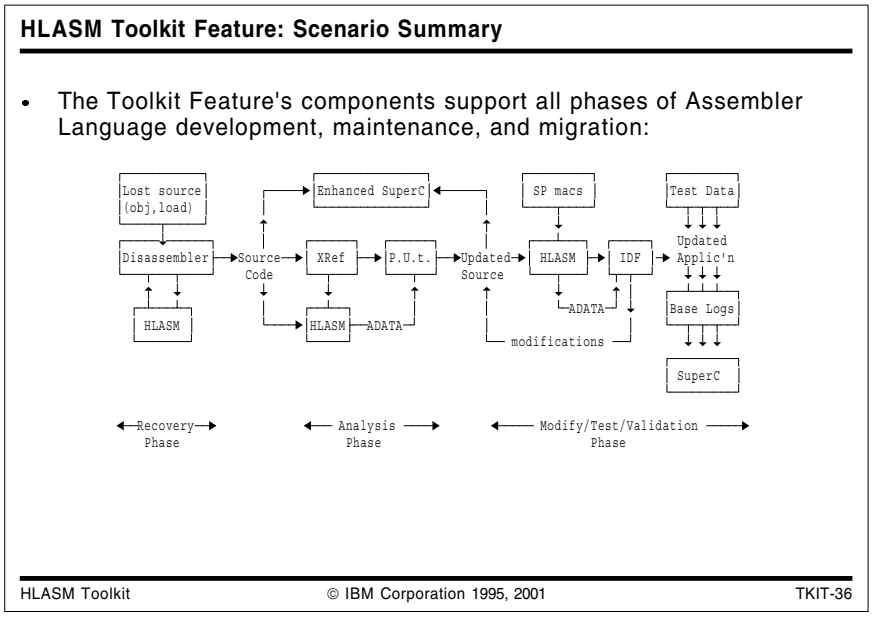
Validation

After some set of modifications has been made to the application, you will probably need to validate its operation. Typical steps in such a process include the following:

1. Run the original unmodified program with a representative set of “old data,” and with the current date set to some manageable “current” date prior to the selected starting date.
2. Run the modified program with the same set of “old data” and the same current date. (There are many techniques available for setting chosen “current” dates on a system.)
3. Use Enhanced SuperC to compare the outputs to ensure that no regressions have been introduced into the existing function of the application. If some date fields have been expanded (such as in report headings), use the date-format facilities of Enhanced SuperC to specify how they should be expanded and compared.

4. Run the modified program with the same “old data” and a new current date.
5. Use Enhanced SuperC to compare these new outputs with the previous two, using Enhanced SuperC's “aging” facilities to ensure correct current-date-dependent behavior.
6. Run the modified program with “new data” and the same new current date.
7. Use Enhanced SuperC to compare the outputs, using “aging” facilities to validate data-dependent behavior.

While not a complete support plan, the above steps are typical of date-sensitive and date-windowing migration and maintenance activities. At each stage, the File Comparison Utility can provide powerful insights into the extent and correctness of code modifications.



These scenarios illustrate how the HLASM Toolkit Feature provides a varied and powerful set of tools supporting all aspects of application development, maintenance, enhancement, and testing. The following figure summarizes these capabilities:

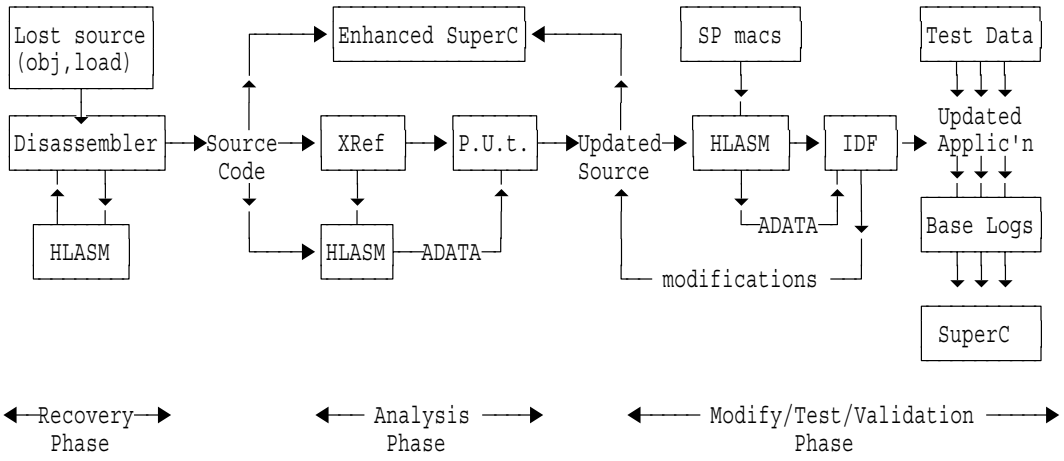
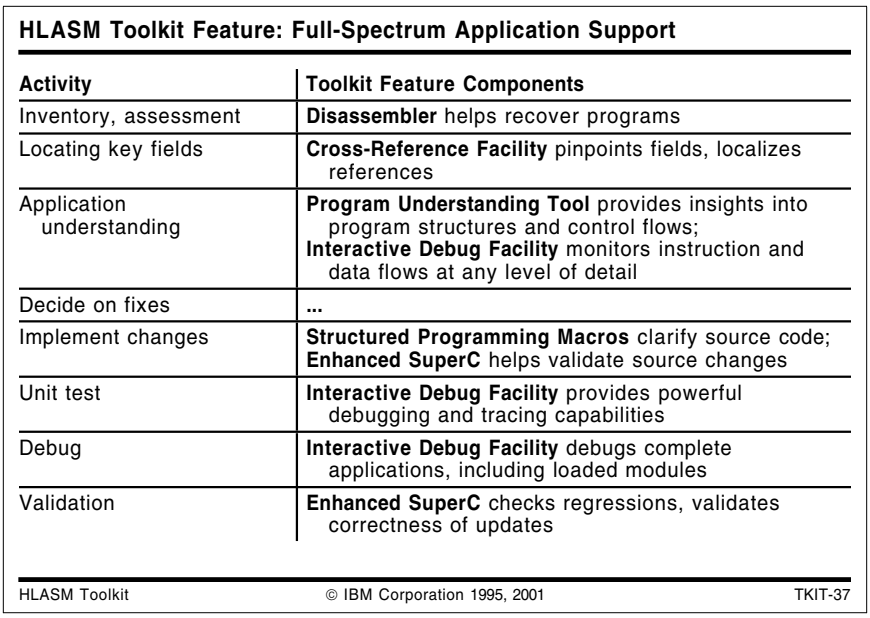


Figure 24. Toolkit Feature: Summary of Usage Scenarios

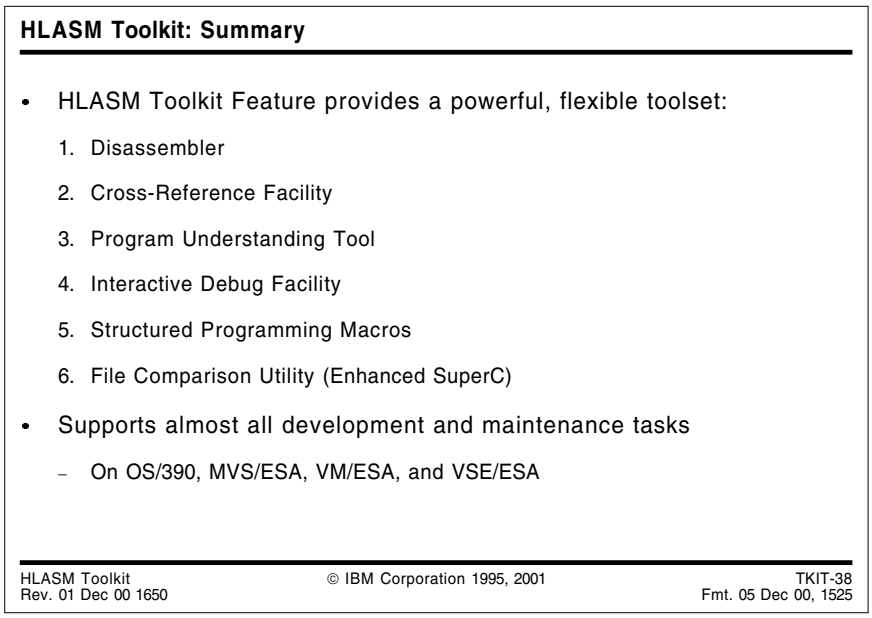


HLASM Toolkit Feature: Full-Spectrum Application Support

A typical process for managing the full spectrum of application recover, development, debugging, and maintenance activities includes several steps. Table 1 shows the Toolkit Feature tools useful in each step.

Table 1. Toolkit Feature Components and Full-Spectrum Application Support

Activity	Toolkit Feature Components
Inventory and assessment	The **Disassembler** can help recover programs previously unretrievable or unmodifiable.
Locating date fields and uses	The **Cross-Reference Facility** pinpoints date fields and localizes references to them.
Application understanding	The **Program Understanding Tool** provides powerful insights into program structures and control flows. The **Interactive Debug Facility** monitors instruction and data flows at any level of detail.
Decide on fixes and methods	
Implement changes	The **Structured Programming Macros** clarify source coding by reducing the need for coding branches and tests, replacing them with readable structures. **Enhanced SuperC** helps verify that source changes are complete.
Unit test	The **Interactive Debug Facility** provides powerful debugging and tracing capabilities for verifying the correctness of changes.
Debug	The **Interactive Debug Facility** helps debug complete applications, including dynamically loaded modules.
Validation	**Enhanced SuperC** checks regressions, validates correctness of updates



Summary

As the preceding examples have shown, the High Level Assembler Toolkit Feature provides a flexible, comprehensive, and powerful set of tools that support many of your application development and maintenance tasks.

Appendix A. Sample structured macro program

```
SAMPLE   CSECT
         BALR  R12,0
         USING *,R12
         IF    AR,R2,R3,NZ,OR,                                    X
               (CLI,WORD1,EQ,X'C1'),ORIF,                         X
               H,AND,                                             X
               CLM,R2,M1,LT,DEC(BASEREG)
           LA  R3,X'01'
           DO  WHILE=(7),UNTIL=(ICM,R2,M1,D2(B2),NZ)
           DOEXIT M
             LA  R3,X'02'
           DOEXIT (CLCL,R2,NL,R4),ANDIF,                          X
               ICM,R2,M1,D2(B2),Z,OR,                             X
               LTR,R2,R3,M,ORIF,                                  X
               10,AND,                                            X
               (CR,R2,NM,R3)
             LA  R3,X'03'
             STRTSRCH  WHILE=(CLM,R2,M1,GE,D2(B2)),UNTIL=P
             EXITIF  Z,AND,                                       X
               LTR,R2,R3,O,ORIF,                                  X
               (CLC,DEC(L,B),EQ,=C'WORD'),AND,                    X
               NP
               LA  R3,X'04'
             ORELSE
               LA  R3,X'05'
             EXITIF  CC=5
               LA  R3,X'06'
             ENDLOOP
               LA  R3,X'07'
             ENDSRCH
           DOEXIT  CC=10
           ENDDO
             LA  R3,X'08'
           IF  (5),ORIF,                                          X
               (CR,R2,NE,R3),ANDIF,                               X
               P,AND,                                             X
               (ICM,R2,M1,D2(B2),O),ORIF,                         X
               CL,R2,LT,D2(B2),OR,                                X
               (LTR,R2,R3,Z)
             LA  R3,X'09'
           ELSE
             LA  R3,X'0A'
           ENDIF
         ENDIF
*
         IF    AR,R2,R3,Z,OR,                                     X
               CR,R3,EQ,R4,AND,                                   X
               AR,R2,R4,NZ,OR,                                    X
               CR,R2,NE,R3
           LA  R3,X'10'
         ENDIF
```

```
*
         IF    AR,R2,R3,Z,AND,                                        X
               CR,R3,EQ,R4,OR,                                        X
               AR,R2,R4,NZ,AND,                                       X
               CR,R2,NE,R3
           LA  R3,X'11'
         ENDIF
*
         IF    AR,R2,R3,Z,ORIF,                                       X
               CR,R3,EQ,R4,ANDIF,                                     X
               AR,R2,R4,NZ,ORIF,                                      X
               CR,R2,NE,R3
           LA  R3,X'12'
         ENDIF
*
         IF    AR,R2,R3,Z,ANDIF,                                      X
               CR,R3,EQ,R4,ORIF,                                      X
               AR,R2,R4,NZ,ANDIF,                                     X
               CR,R2,NE,R3
           LA  R3,X'13'
         ENDIF
*
         DO    INF
           LA  R5,X'00'
         DOEXIT  (O)
           LA  R5,X'05'
           DO  FROM=(R3,10),TO=(R4,-1)
             LA  R5,X'0A'
           DOEXIT  (CR,R3,E,R5)
             LA  R5,X'0F'
             DO  WHILE=(NR,R2,R4,Z)
               LA  R5,X'10'
             DOEXIT  (NZ)
               LA  R5,X'15'
             ENDDO
             LA  R5,X'1A'
           ENDDO
           LA  R5,X'1F'
         ENDDO
*
         CASENTRY  R4,POWER=3
         CASE  (16,32)
           LA  R4,X'03'
           CASENTRY  R4,VECTOR=BR
             LA  R2,X'10'
           CASE  1,3,4
             LA  R3,X'20'
           CASE  5
             LA  R3,X'30'
           ENDCASE
           CASENTRY  R4,POWER=1,VECTOR=B
           CASE  4
             LA  R3,X'40'
           CASE (2)
             LA  R3,X'50'
           ENDCASE
```

```
         CASE  24
           CASENTRY  R4,POWER=0
           CASE  1
             SELECT  C,R2,GE
               WHEN (=F'100')
                 LA   R0,3
               WHEN (=F'1000')
                 LA   R0,4
               WHEN (=F'10000')
                 LA   R0,5
               OTHRWISE
                 LA   R0,10
               ENDSEL
             LA  R3,X'60'
           CASE  2
             LA  R3,X'70'
           ENDCASE
         ENDCASE
*
         SELECT  CLC,WORD1,EQ
           WHEN  (=C'+',=C'-')
             LA    R0,1
           WHEN  (=C'*',=C'/')
             LA    R0,2
           OTHRWISE
             SR    R0,R0
             CASENTRY  R4,POWER=2
             CASE  8
               LA  R3,X'80'
             ENDCASE
           ENDSEL
*
WORD1    DC    CL1'A'
B        EQU   4
BASEREG  EQU   5
B2       EQU   6
DEC      EQU   16
D2       EQU   32
L        EQU   64
LENGTH   EQU   4
M1       EQU   6
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R12      EQU   12
         END   SAMPLE
```

Appendix B. Listing of sample program

```
                                    High Level Assembler Option Summary                 (PTF UN96931)   Page    1
                                                                                       HLASM R2.0  1997/02/03 15.17
Overriding Parameters-  profile(asmmsp),list(121),noesd,norld,noxref,nomxref,nousing
No Process Statements


Options for this Assembly
 NOADATA
   ALIGN
   ASA
   BATCH
 NOCOMPAT
 NODBCS
 NODECK
   DXREF
 NOESD
 NOEXIT
   FLAG(0,ALIGN,CONT,RECORD,NOSUBSTR)
 NOFOLD
   LANGUAGE(EN)
 NOLIBMAC
   LINECOUNT(59)
   LIST(121)
 NOMXREF
 NOOBJECT
   OPTABLE(UNI)
 NOPCONTROL
 NOPESTOP
   PROFILE(ASMMSP)
 NORA2
 NORENT
 NORLD
   SIZE(MAX)
   SYSPARM()
   TERM(NARROW)
 NOTEST
 NOTRANSLATE
 NOUSING
 NOXOBJECT
 NOXREF

No Overriding DD Names
                                                                                                          Page    2
Active Usings: None
Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                 HLASM R2.0  1997/02/03 15.17
                                   1         COPY ASMMSP   Generated for PROFILE option
000000                             955 SAMPLE   CSECT
000000 05C0                        956          BALR  R12,0
                R:C  00002         957          USING *,R12
                                   958          IF    AR,R2,R3,NZ,OR,                                      X
                                                      (CLI,WORD1,EQ,X'C1'),ORIF,                           X
                                                      H,AND,                                               X
                                                      CLM,R2,M1,LT,DEC(BASEREG)
000002 1A23                        959+               AR         R2,R3                                      03-ASMMP
000004 4770 C01A            0001C  960+               BC         7,#@LB2                                    03-ASMMP
000008 95C1 C2D2      002D4        961+               CLI        WORD1,X'C1'                                03-ASMMP
00000C 4780 C01A            0001C  962+               BC         8,#@LB2                                    03-ASMMP
000010 47D0 C0CC            000CE  963+               BC         15-2,#@LB1                                 03-ASMMP
000014 BD26 5010            00010  964+               CLM        R2,M1,DEC(BASEREG)                         03-ASMMP
000018 47B0 C0CC            000CE  965+             BC 15-4,#@LB1                                          02-ASMMI
                            0001C  966+#@LB2          EQU *                                              02-ASMMI
00001C 4130 0001            00001  967            LA  R3,X'01'
                                   968            DO  WHILE=(7),UNTIL=(ICM,R2,M1,D2(B2),NZ)
000020 4780 C098            0009A  969+#@LB4           BC    15-7,#@LB3                                   02-ASMMD
                                   970            DOEXIT M
000024 4740 C098            0009A  971+            BC    4,#@LB3                                        02-ASMMI
000028 4130 0002            00002  972              LA  R3,X'02'
                                   973            DOEXIT (CLCL,R2,NL,R4),ANDIF,                        X
                                                      ICM,R2,M1,D2(B2),Z,OR,                               X
```

```
                                                  LTR,R2,R3,M,ORIF,                                      X
                                                  10,AND,                                                X
                                                  (CR,R2,NM,R3)
00002C 0F24                        974+           CLCL       R2,R4                                        03-ASMMP
00002E 4740 C03E            00040  975+           BC         15-11,#@LB7                                  03-ASMMP
000032 BF26 6020            00020  976+           ICM        R2,M1,D2(B2)                                 03-ASMMP
000036 4780 C098            0009A  977+           BC         8,#@LB3                                      03-ASMMP
00003A 1223                        978+           LTR        R2,R3                                        03-ASMMP
00003C 4740 C098            0009A  979+           BC         4,#@LB3                                      03-ASMMP
                            00040  980+#@LB7       EQU        *                                            03-ASMMP
000040 4750 C048            0004A  981+           BC         15-10,#@LB9                                  03-ASMMP
000044 1923                        982+           CR         R2,R3                                        03-ASMMP
000046 47B0 C098            0009A  983+         BC    11,#@LB3                                            02-ASMMI
                            0004A  984+#@LB9      EQU  *                                                 02-ASMMI
00004A 4130 0003            00003  985           LA  R3,X'03'
                                   986           STRTSRCH  WHILE=(CLM,R2,M1,GE,D2(B2)),UNTIL=P
00004E BD26 6020            00020  987+#@LB12      CLM        R2,M1,D2(B2)                                 03-ASMMP
000052 4740 C088            0008A  988+           BC    15-11,#@LB11                                     02-ASMMD
                                   989           EXITIF  Z,AND,                                         X
                                                   LTR,R2,R3,O,ORIF,                                    X
                                                   (CLC,DEC(L,B),EQ,=C'WORD'),AND,                      X
                                                   NP
000056 4770 C05E            00060  990+           BC         15-8,#@LB13                                  03-ASMMP
00005A 1223                        991+           LTR        R2,R3                                        03-ASMMP
00005C 4710 C06C            0006E  992+           BC         1,#@LB14                                     03-ASMMP
                            00060  993+#@LB13      EQU        *                                            03-ASMMP
000060 D53F 4010 C2D6 00010 002D8  994+           CLC        DEC(L,B),=C'WORD'                            03-ASMMP
000066 4770 C074            00076  995+           BC         15-8,#@LB15                                  03-ASMMP
00006A 4720 C074            00076  996+         BC 15-13,#@LB15                                        02-ASMMI
                            0006E  997+#@LB14      EQU *                                                 02-ASMMI
                                                                                                         Page    3
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM R2.0  1997/02/03 15.17
00006E 4130 0004            00004   998             LA  R3,X'04'
                                    999           ORELSE
000072 47F0 C08C            0008E  1000+           BC    15,#@LB10                                        01-00702
                            00076  1001+#@LB15     EQU *                                                  01-00703
000076 4130 0005            00005  1002              LA  R3,X'05'
                                   1003           EXITIF  CC=5
00007A 47A0 C084            00086  1004+           BC 15-5,#@LB16                                       02-ASMMI
00007E 4130 0006            00006  1005              LA  R3,X'06'
                                   1006           ENDLOOP
000082 47F0 C08C            0008E  1007+         BC    15,#@LB10                                        01-00527
                            00086  1008+#@LB16    EQU   *                                                01-00528
000086 47D0 C04C            0004E  1009+             BC         15-2,#@LB12                              02-ASMMP
                            0008A  1010+#@LB11     EQU         *                                         02-ASMMP
00008A 4130 0007            00007  1011              LA  R3,X'07'
                                   1012           ENDSRCH
                            0008E  1013+#@LB10     EQU   *                                                01-00641
                                   1014          DOEXIT  CC=10
00008E 47A0 C098            0009A  1015+          BC    10,#@LB3                                         02-ASMMI
                                   1016          ENDDO
000092 BF26 6020            00020  1017+             ICM        R2,M1,D2(B2)                             02-ASMMP
000096 4780 C01E            00020  1018+             BC         15-7,#@LB4                               02-ASMMP
                            0009A  1019+#@LB3      EQU         *                                         02-ASMMP
00009A 4130 0008            00008  1020            LA  R3,X'08'
                                   1021          IF  (5),ORIF,                                          X
                                                   (CR,R2,NE,R3),ANDIF,                                 X
                                                   P,AND,                                               X
                                                   (ICM,R2,M1,D2(B2),O),ORIF,                           X
                                                   CL,R2,LT,D2(B2),OR,                                  X
                                                   (LTR,R2,R3,Z)
00009E 4750 C0A6            000A8  1022+           BC         5,#@LB21                                 03-ASMMP
0000A2 1923                        1023+          CR         R2,R3                                    03-ASMMP
0000A4 4780 C0B2            000B4  1024+          BC         15-7,#@LB20                              03-ASMMP
                            000A8  1025+#@LB21     EQU *                                               02-ASMMI
0000A8 47D0 C0B2            000B4  1026+          BC         15-2,#@LB20                              03-ASMMP
0000AC BF26 6020            00020  1027+          ICM        R2,M1,D2(B2)                             03-ASMMP
0000B0 4710 C0C0            000C2  1028+          BC         1,#@LB22                                 03-ASMMP
                            000B4  1029+#@LB20     EQU        *                                        03-ASMMP
0000B4 5526 0020            00020  1030+          CL         R2,D2(B2)                                03-ASMMP
0000B8 4740 C0C0            000C2  1031+          BC         4,#@LB22                                 03-ASMMP
0000BC 1223                        1032+          LTR        R2,R3                                    03-ASMMP
0000BE 4770 C0C8            000CA  1033+        BC 15-8,#@LB23                                         02-ASMMI
```

```
                         000C2  1034+#@LB22         EQU *                                                  02-ASMMI
0000C2 4130 0009         00009  1035              LA  R3,X'09'
                                1036            ELSE
0000C6 47F0 C0CC         000CE  1037+              BC   15,#@LB24                                          01-00287
                         000CA  1038+#@LB23        EQU  *                                                 01-00289
0000CA 4130 000A         0000A  1039              LA  R3,X'0A'
                                1040            ENDIF
                         000CE  1041+#@LB24       EQU *                                                  01-00478
                                1042          ENDIF
                         000CE  1043+#@LB1        EQU *                                                  01-00478
                                1044 *
                                1045          IF    AR,R2,R3,Z,OR,                                     X
                                                    CR,R3,EQ,R4,AND,                                   X
                                                                                                       Page    4
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                              HLASM R2.0  1997/02/03 15.17
                                                    AR,R2,R4,NZ,OR,                                    X
                                                    CR,R2,NE,R3
0000CE 1A23                     1046+               AR         R2,R3                                    03-ASMMP
0000D0 4780 C0E4         000E6  1047+               BC         8,#@LB26                                 03-ASMMP
0000D4 1934                     1048+               CR         R3,R4                                    03-ASMMP
0000D6 4770 C0E8         000EA  1049+               BC         15-8,#@LB25                              03-ASMMP
0000DA 1A24                     1050+               AR         R2,R4                                    03-ASMMP
0000DC 4770 C0E4         000E6  1051+               BC         7,#@LB26                                 03-ASMMP
0000E0 1923                     1052+               CR         R2,R3                                    03-ASMMP
0000E2 4780 C0E8         000EA  1053+             BC 15-7,#@LB25                                       02-ASMMI
                         000E6  1054+#@LB26         EQU *                                              02-ASMMI
0000E6 4130 0010         00010  1055            LA  R3,X'10'
                                1056          ENDIF
                         000EA  1057+#@LB25       EQU *                                                  01-00478
                                1058 *
                                1059          IF    AR,R2,R3,Z,AND,                                    X
                                                    CR,R3,EQ,R4,OR,                                    X
                                                    AR,R2,R4,NZ,AND,                                   X
                                                    CR,R2,NE,R3
0000EA 1A23                     1060+               AR         R2,R3                                    03-ASMMP
0000EC 4770 C104         00106  1061+               BC         15-8,#@LB27                              03-ASMMP
0000F0 1934                     1062+               CR         R3,R4                                    03-ASMMP
0000F2 4780 C100         00102  1063+               BC         8,#@LB28                                 03-ASMMP
0000F6 1A24                     1064+               AR         R2,R4                                    03-ASMMP
0000F8 4780 C104         00106  1065+               BC         15-7,#@LB27                              03-ASMMP
0000FC 1923                     1066+               CR         R2,R3                                    03-ASMMP
0000FE 4780 C104         00106  1067+             BC 15-7,#@LB27                                       02-ASMMI
                         00102  1068+#@LB28         EQU *                                              02-ASMMI
000102 4130 0011         00011  1069            LA  R3,X'11'
                                1070          ENDIF
                         00106  1071+#@LB27       EQU *                                                  01-00478
                                1072 *
                                1073          IF    AR,R2,R3,Z,ORIF,                                   X
                                                    CR,R3,EQ,R4,ANDIF,                                 X
                                                    AR,R2,R4,NZ,ORIF,                                  X
                                                    CR,R2,NE,R3
000106 1A23                     1074+               AR         R2,R3                                    03-ASMMP
000108 4780 C110         00112  1075+               BC         8,#@LB30                                 03-ASMMP
00010C 1934                     1076+               CR         R3,R4                                    03-ASMMP
00010E 4770 C116         00118  1077+               BC         15-8,#@LB29                              03-ASMMP
                         00112  1078+#@LB30         EQU *                                              02-ASMMI
000112 1A24                     1079+               AR         R2,R4                                    03-ASMMP
000114 4770 C11C         0011E  1080+               BC         7,#@LB31                                 03-ASMMP
                         00118  1081+#@LB29         EQU        *                                       03-ASMMP
000118 1923                     1082+               CR         R2,R3                                    03-ASMMP
00011A 4780 C120         00122  1083+             BC 15-7,#@LB32                                       02-ASMMI
                         0011E  1084+#@LB31         EQU *                                              02-ASMMI
00011E 4130 0012         00012  1085            LA  R3,X'12'
                                1086          ENDIF
                         00122  1087+#@LB32       EQU *                                                  01-00478
                                1088 *
                                1089          IF    AR,R2,R3,Z,ANDIF,                                  X
                                                    CR,R3,EQ,R4,ORIF,                                  X
                                                    AR,R2,R4,NZ,ANDIF,                                 X
                                                                                                       Page    5
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                              HLASM R2.0  1997/02/03 15.17
```

```
                                                CR,R2,NE,R3
000122 1A23                    1090+               AR        R2,R3                                  03-ASMMP
000124 4770 C12C        0012E  1091+               BC        15-8,#@LB33                            03-ASMMP
000128 1934                    1092+               CR        R3,R4                                  03-ASMMP
00012A 4780 C132        00134  1093+               BC        8,#@LB34                               03-ASMMP
                        0012E  1094+#@LB33         EQU       *                                      03-ASMMP
00012E 1A24                    1095+               AR        R2,R4                                  03-ASMMP
000130 4780 C13C        0013E  1096+               BC        15-7,#@LB35                            03-ASMMP
                        00134  1097+#@LB34         EQU *                                            02-ASMMI
000134 1923                    1098+               CR        R2,R3                                  03-ASMMP
000136 4780 C13C        0013E  1099+             BC 15-7,#@LB35                                     02-ASMMI
00013A 4130 0013        00013  1100              LA  R3,X'13'
                               1101            ENDIF
                        0013E  1102+#@LB35       EQU *                                            01-00478
                               1103 *
                               1104            DO    INF
                        0013E  1105+#@LB38           EQU *                                        02-ASMMD
00013E 4150 0000        00000  1106              LA  R5,X'00'
                               1107            DOEXIT  (0)
000142 4710 C184        00186  1108+             BC    1,#@LB37                                   02-ASMMI
000146 4150 0005        00005  1109              LA  R5,X'05'
                               1110              DO  FROM=(R3,10),TO=(R4,-1)
00014A 4130 000A        0000A  1111+                   LA  R3,10                                  02-ASMMD
00014E 4840 C2E6        002E8  1112+                 LH  R4,=H'-1'                                02-ASMMD
                        00152  1113+#@LB42      EQU   *                                          02-ASMMD
000152 4150 000A        0000A  1114                LA  R5,X'0A'
                               1115              DOEXIT  (CR,R3,E,R5)
000156 1935                    1116+                CR        R3,R5                                 03-ASMMP
000158 4780 C17C        0017E  1117+             BC    8,#@LB41                                   02-ASMMI
00015C 4150 000F        0000F  1118                LA  R5,X'0F'
                               1119              DO  WHILE=(NR,R2,R4,Z)
000160 47F0 C16E        00170  1120+                 BC    15,#@LB46                              02-ASMMD
                        00164  1121+#@LB47            EQU  *                                     02-ASMMD
000164 4150 0010        00010  1122                  LA  R5,X'10'
                               1123              DOEXIT  (NZ)
000168 4770 C174        00176  1124+             BC    7,#@LB45                                   02-ASMMI
00016C 4150 0015        00015  1125                  LA  R5,X'15'
                               1126              ENDDO
000170 1424                    1127+#@LB46           NR        R2,R4                                02-ASMMP
000172 4780 C162        00164  1128+                 BC        8,#@LB47                             02-ASMMP
                        00176  1129+#@LB45       EQU         *                                    02-ASMMP
000176 4150 001A        0001A  1130                LA  R5,X'1A'
                               1131              ENDDO
00017A 8734 C150        00152  1132+                 BXLE      R3,R4,#@LB42                         02-ASMMP
                        0017E  1133+#@LB41       EQU         *                                    02-ASMMP
00017E 4150 001F        0001F  1134              LA  R5,X'1F'
                               1135            ENDDO
000182 47F0 C13C        0013E  1136+                 BC        15,#@LB38                            02-ASMMP
                        00186  1137+#@LB37       EQU         *                                    02-ASMMP
                               1138 *
                               1139            CASENTRY  R4,POWER=3
000186 8A40 0001        00001  1140+                SRA R4,3-2                                     01-00167
00018A 5A40 C192        00194  1141+                A   R4,#@LB52                                 01-00172
00018E 5844 0000        00000  1142+                L   R4,0(R4)                                 01-00173
                                                                                                  Page    6
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                          HLASM R2.0  1997/02/03 15.17
000192 07F4                    1143+                BCR 15,R4                                      01-00174
000194 00000260                1144+#@LB52          DC  A(#@LB50)                                  01-00175
                               1145            CASE  (16,32)
                        00198  1146+#@LB53          EQU *                                        01-00096
000198 4140 0003        00003  1147              LA  R4,X'03'
                               1148              CASENTRY  R4,VECTOR=BR
00019C 8B40 0002        00002  1149+                 SLA R4,2-0                                  01-00165
0001A0 47F4 C1B2        001B4  1150+               BC  15,#@LB54(R4)                              01-00178
0001A4 4120 0010        00010  1151                 LA  R2,X'10'
                               1152              CASE  1,3,4
                        001A8  1153+#@LB56           EQU *                                        01-00096
0001A8 4130 0020        00020  1154                 LA  R3,X'20'
                               1155              CASE  5
0001AC 47F0 C1CA        001CC  1156+                 BC    15,#@LB55                              01-00094
                        001B0  1157+#@LB57           EQU *                                        01-00096
0001B0 4130 0030        00030  1158                 LA  R3,X'30'
```

```
                                1159           ENDCASE
0001B4 47F0 C1CA           001CC 1160+#@LB54        BC    15,#@LB55                                  01-00344
0001B8 47F0 C1A6           001A8 1161+              BC    15,#@LB56                                  01-00359
0001BC 47F0 C1CA           001CC 1162+              BC    15,#@LB55                                  01-00369
0001C0 47F0 C1A6           001A8 1163+              BC    15,#@LB56                                  01-00359
0001C4 47F0 C1A6           001A8 1164+              BC    15,#@LB56                                  01-00359
0001C8 47F0 C1AE           001B0 1165+              BC    15,#@LB57                                  01-00359
                           001CC 1166+#@LB55          EQU   *                                        01-00374
                                 1167           CASENTRY  R4,POWER=1,VECTOR=B
0001CC 8B40 0001           00001 1168+              SLA R4,2-1                                       01-00165
0001D0 47F4 C1DE           001E0 1169+            BC  15,#@LB58(R4)                                  01-00178
                                 1170           CASE  4
                           001D4 1171+#@LB60        EQU *                                          01-00096
0001D4 4130 0040           00040 1172             LA  R3,X'40'
                                 1173           CASE (2)
0001D8 47F0 C1EA           001EC 1174+            BC    15,#@LB59                                  01-00094
                           001DC 1175+#@LB61      EQU *                                            01-00096
0001DC 4130 0050           00050 1176             LA  R3,X'50'
                                 1177           ENDCASE
0001E0 47F0 C1EA           001EC 1178+#@LB58        BC    15,#@LB59                                  01-00344
0001E4 47F0 C1DA           001DC 1179+              BC    15,#@LB61                                  01-00359
0001E8 47F0 C1D2           001D4 1180+              BC    15,#@LB60                                  01-00359
                           001EC 1181+#@LB59          EQU   *                                        01-00374
                                 1182           CASE  24
0001EC 5840 C25E           00260 1183+              L     R4,#@LB50                                  01-00091
0001F0 07F4                      1184+              BCR   15,R4                                      01-00092
                           001F2 1185+#@LB62        EQU *                                          01-00096
                                 1186             CASENTRY  R4,POWER=0
0001F2 8B40 0002           00002 1187+              SLA R4,2-0                                       01-00165
0001F6 5A40 C1FE           00200 1188+            A   R4,#@LB65                                    01-00172
0001FA 5844 0000           00000 1189+            L   R4,0(R4)                                     01-00173
0001FE 07F4                      1190+            BCR 15,R4                                        01-00174
000200 0000024C                  1191+#@LB65      DC  A(#@LB63)                                    01-00175
                                 1192           CASE  1
                           00204 1193+#@LB66      EQU *                                            01-00096
                                 1194           SELECT  C,R2,GE
                                 1195             WHEN (=F'100')
000204 5920 C2DA           002DC 1196+        C            R2,=F'100'                                01-00931
                                                                                                   Page    7
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                              HLASM R2.0  1997/02/03 15.17
000208 4740 C212           00214 1197+        BC    15-11,#@LB68                                     01-00939
00020C 4100 0003           00003 1198                 LA   R0,3
                                 1199           WHEN (=F'1000')
000210 47F0 C236           00238 1200+        B     #@LB67            SKIP TO END                    01-00908
                           00214 1201+#@LB68        EQU *                                          01-00911
000214 5920 C2DE           002E0 1202+        C            R2,=F'1000'                               01-00931
000218 4740 C222           00224 1203+        BC    15-11,#@LB70                                     01-00939
00021C 4100 0004           00004 1204                 LA   R0,4
                                 1205           WHEN (=F'10000')
000220 47F0 C236           00238 1206+        B     #@LB67            SKIP TO END                    01-00908
                           00224 1207+#@LB70        EQU *                                          01-00911
000224 5920 C2E2           002E4 1208+        C            R2,=F'10000'                              01-00931
000228 4740 C232           00234 1209+        BC    15-11,#@LB72                                     01-00939
00022C 4100 0005           00005 1210                 LA   R0,5
                                 1211           OTHRWISE
000230 47F0 C236           00238 1212+        B     #@LB67            SKIP TO END                    01-00755
                           00234 1213+#@LB72        EQU *                                          01-00757
000234 4100 000A           0000A 1214                 LA   R0,10
                                 1215           ENDSEL
                           00238 1216+#@LB67         EQU *                                         01-00591
000238 4130 0060           00060 1217             LA  R3,X'60'
                                 1218           CASE  2
00023C 5840 C24A           0024C 1219+              L     R4,#@LB63                                  01-00091
000240 07F4                      1220+              BCR   15,R4                                      01-00092
                           00242 1221+#@LB74        EQU *                                          01-00096
000242 4130 0070           00070 1222             LA  R3,X'70'
                                 1223           ENDCASE
000246 5840 C24A           0024C 1224+              L     R4,#@LB63                                  01-00339
00024A 07F4                      1225+              BCR   15,R4                                      01-00340
00024C 00000258                  1226+#@LB63        DC    A(#@LB64)                                  01-00341
000250 00000204                  1227+                DC    A(#@LB66)                                01-00357
000254 00000242                  1228+                DC    A(#@LB74)                                01-00357
                           00258 1229+#@LB64          EQU   *                                        01-00374
```

```
                                1230          ENDCASE
000258 5840 C25E           00260 1231+               L     R4,#@LB50                                       01-00339
00025C 07F4                      1232+               BCR   15,R4                                           01-00340
00025E 0000                         +
000260 00000274                  1233+#@LB50         DC    A(#@LB51)                                       01-00341
000264 00000274                  1234+                 DC    A(#@LB51)                                       01-00354
000268 00000198                  1235+                 DC    A(#@LB53)                                       01-00357
00026C 000001F2                  1236+                 DC    A(#@LB62)                                       01-00357
000270 00000198                  1237+                 DC    A(#@LB53)                                       01-00357
                           00274 1238+#@LB51           EQU   *                                               01-00374
                                 1239 *
                                 1240          SELECT  CLC,WORD1,EQ
                                 1241            WHEN  (=C'+',=C'-')
000274 D500 C2D2 C2E8 002D4 002EA 1242+         CLC           WORD1,=C'+'                                     01-00931
00027A 4780 C286           00288 1243+         BC    8,#@LB77                                                01-00935
00027E D500 C2D2 C2E9 002D4 002EB 1244+         CLC           WORD1,=C'-'                                     01-00931
000284 4770 C28E           00290 1245+         BC    15-8,#@LB76                                             01-00939
                           00288 1246+#@LB77     EQU  *                                                      01-00942
000288 4100 0001           00001 1247              LA    R0,1
                                 1248            WHEN  (=C'*',=C'/')
00028C 47F0 C2D2           002D4 1249+         B     #@LB75             SKIP TO END                          01-00908
                                                                                                          Page    8
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                 HLASM R2.0  1997/02/03 15.17
                           00290 1250+#@LB76         EQU *                                                 01-00911
000290 D500 C2D2 C2EA 002D4 002EC 1251+         CLC           WORD1,=C'*'                                     01-00931
000296 4780 C2A2           002A4 1252+         BC    8,#@LB79                                                01-00935
00029A D500 C2D2 C2EB 002D4 002ED 1253+         CLC           WORD1,=C'/'                                     01-00931
0002A0 4770 C2AA           002AC 1254+         BC    15-8,#@LB78                                             01-00939
                           002A4 1255+#@LB79     EQU  *                                                      01-00942
0002A4 4100 0002           00002 1256              LA    R0,2
                                 1257            OTHRWISE
0002A8 47F0 C2D2           002D4 1258+         B     #@LB75             SKIP TO END                          01-00755
                           002AC 1259+#@LB78         EQU *                                                 01-00757
0002AC 1B00                      1260              SR    R0,R0
                                 1261              CASENTRY  R4,POWER=2
0002AE 5A40 C2B6           002B8 1262+             A   R4,#@LB82                                           01-00172
0002B2 5844 0000           00000 1263+             L   R4,0(R4)                                            01-00173
0002B6 07F4                      1264+             BCR 15,R4                                               01-00174
0002B8 000002C8                  1265+#@LB82       DC  A(#@LB80)                                           01-00175
                                 1266              CASE  8
                           002BC 1267+#@LB83       EQU *                                                   01-00096
0002BC 4130 0080           00080 1268                LA  R3,X'80'
                                 1269              ENDCASE
0002C0 5840 C2C6           002C8 1270+               L     R4,#@LB80                                       01-00339
0002C4 07F4                      1271+               BCR   15,R4                                           01-00340
0002C6 0000                         +
0002C8 000002D4                  1272+#@LB80         DC    A(#@LB81)                                       01-00341
0002CC 000002D4                  1273+                 DC    A(#@LB81)                                       01-00354
0002D0 000002BC                  1274+                 DC    A(#@LB83)                                       01-00357
                           002D4 1275+#@LB81           EQU   *                                               01-00374
                                 1276            ENDSEL
                           002D4 1277+#@LB75          EQU *                                                01-00591
                                 1278 *
0002D4 C1                        1279 WORD1    DC    CL1'A'
                           00004 1280 B        EQU   4
                           00005 1281 BASEREG  EQU   5
                           00006 1282 B2       EQU   6
                           00010 1283 DEC      EQU   16
                           00020 1284 D2       EQU   32
                           00040 1285 L        EQU   64
                           00004 1286 LENGTH   EQU   4
                           00006 1287 M1       EQU   6
                           00000 1288 R0       EQU   0
                           00001 1289 R1       EQU   1
                           00002 1290 R2       EQU   2
                           00003 1291 R3       EQU   3
                           00004 1292 R4       EQU   4
                           00005 1293 R5       EQU   5
                           00006 1294 R6       EQU   6
                           0000C 1295 R12      EQU   12
000000                           1296          END   SAMPLE
0002D8 E6D6D9C4                  1297                =C'WORD'
0002DC 00000064                  1298                =F'100'
```

```
0002E0 000003E8                  1299               =F'1000'
0002E4 00002710                  1300               =F'10000'
0002E8 FFFF                      1301               =H'-1'
0002EA 4E                        1302               =C'+'
                                                                                                    Page    9
  Active Usings: SAMPLE+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM R2.0  1997/02/03 15.17
0002EB 60                        1303               =C'-'
0002EC 5C                        1304               =C'*'
0002ED 61                        1305               =C'/'
                                Diagnostic Cross Reference and Assembler Summary                    Page   10
                                                                                    HLASM R2.0  1997/02/03 15.17
     No Statements Flagged in this Assembly
HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 2.0, PTF UN96931
SYSTEM: CMS 11                   JOBNAME: (NOJOB)     STEPNAME: (NOSTEP)   PROCSTEP: (NOPROC)
Datasets Allocated for this Assembly
 Con DDname   Dataset Name                              Volume  Member
  P1 SYSIN    SAMPLE   ASSEMBLE D1                      EHR192
  L1 SYSLIB   ASMAMAC  MACLIB   L1                      EHR199
  L2          ASMSMAC  MACLIB   L2                      EHR199
  L3          OSMACRO  MACLIB   S2                      RB-190
     SYSPRINT SAMPLE   LISTING  D1                      EHR192
     SYSTERM  TERMINAL

   2759K allocated to Buffer Pool,          245K would be required for this to be an In-Storage Assembly
    157 Primary Input Records Read         1861 Library Records Read          0 Work File Reads
    467 Primary Print Records Written         0 Punch Records Written         0 Work File Writes
      0 ADATA Records Written
Assembly Start Time: 15.17.01 Stop Time: 15.17.01 Processor Time: 00.00.00.5245
Return Code 000
```