Systems/ASM
Version 1.95

Systems/ASM
Version 1.95

Contents

How to use this book

This book describes the Dignus Systems/ASM assembler, **DASM**. **DASM** is used to assemble mainframe assembly source code, producing mainframe object files. This book does not describe S/390 or zSeries assembler language programming in detail. For more information regarding S/390 or zSeries programming, please see the IBM book *High Level Assembler (HLASM) for MVS & VM & VSE V1R6 Language Reference* or other IBM materials.

For further information, contact Dignus, LLC at (919) 676-0847, or visit `http://www.dignus.com`.

The Systems/ASM Assembler
DASM

Systems/ASM Overview

The Dignus Systems/ASM assembler, **DASM**, is an assembler compatible with IBM's HLASM assembler for the mainframe family of machines.

Some of its features include:

- HLASM V1R6 compatible assembler.
- 'asma90' compatibility for operating similarly to IBM asma90.
- Available as a cross-platform assembler on several cross-platform hosts.
- Generates dependency information, detailing which files were involved in an assembly.
- Supports IBM defined instructions up through the z10 architecture.
- Support for IBM 'events' files for integration with other IBM development products.
- Support for building Linux/390, z/Linux and z/TPF programs.
- Enhanced HTML listing.
- Compatible with the Systems/C and Systems/C++ compilers.

DASM Advanced Features and Extensions

The Dignus Assembler, **DASM**, provides many advanced features. These features combine to produce a programming environment which is perfectly suited for many programming tasks.

HLASM V1R6 compatibility

DASM is compatible with HLASM V1R6, including unsigned constants, newer instruction formats and instructions, Decimal data, the QY and SY constant formats, mnemonic suffixes and other new HLASM V1R6 features.

XSD support

When the *–xsd* option is enabled, the assembler will generate `XSD` cards in the object deck instead of `ESD` cards. `XSD` cards allow for names longer than the typical 8 characters.

For those environments where the binder is not supported, the Systems/C pre-linker **PLINK** can be used to process the objects and appropriately shorten the names. For more information about **PLINK**, see the *Systems/C Utilities* manual.

GOFF support

When the *–goff* option is enabled, the assembler will generate GOFF-style object decks. Similar to XSD-style, GOFF-style object decks allow for names longer than 8 characters and have other features.

Both **PLINK** and IBM's binder supports GOFF-style object decks. For those environments where the binder is not supported (e.g. VSE), the Systems/C pre-linker **PLINK** can then be used to process these GOFF objects and appropriately

shorten the names and produce old-style ESD object decks. For more information about **PLINK**, see the *Systems/C Utilities* manual.

Linux ELF Output

When the *–flinux* or the *–flinux64* option is used, **DASM** generates ELF (`.o` file) output that is compatible with the Linux linker (**ld**) and run-time. The input source is then HLASM compatible syntax. See the chapter "Writing Linux/390 and z/Linux programs" for details.

ADATA information

DASM will also optionally produce Associated Data, or ADATA information. ADATA information describes the source being assembled, as well as any user-provided information that appears in `ADATA` statements. ADATA information can either be written to a separate file, or embedded in a GOFF-format file.

DASM can generate either HLASM V1R5 format or HLASM V1R4 format ADATA information.

z/Architecture Instructions and Data

DASM supports the newer z/Architecture instructions and data formats, including the z/Architecture instruction and data formats, 20-bit offsets, newer relocation types and decimal instructions and data types.

Cross-Platform support

As well as z/OS, **DASM** is available on several cross-platform hosts; allowing for the assembly of HLASM compatible source on a UNIX or Windows workstation.

Dependency List generation

The assembler, using the *–M* option, can generate a list of the macros and copy members used by an assembler source. This feature can be employed to automatically generate build dependencies for constructing complex programs.

HTML listing format

The assembler, using the *–flisting_style=html* option, produces the listing using the HTML markup language. This listing can be viewed with a web browser. Listings in this format have links between the cross-reference and messages back to the lines in the listing as well as other features that make reading it easier than traditional listings.

Systems/C and Systems/C++ Integration

DASM has several opcode extensions which make it a perfect companion for the Dignus compiler products.

Assembling Programs

This section describes the Dignus Systems/ASM assembler, **DASM**, explaining how to assemble the assembly language source, how to link modules to build an executable and how to run the resulting program. It is not intended to be a complete description of HLASM or mainframe assembly language programming; for that, please consult other texts such as IBM's *High Level Assembler (HLASM) for MVS & VM & VSE V1R6 Language Reference*.

This chapter explains how to run the **DASM** assembler and what options are available on **DASM**.

Accessing files on z/OS

The IBM-supplied system macros and copy members are available on the z/OS platform, and can readily be used by **DASM**. **DASM** can directly reference PDS members while running in an OpenEdition (USS) environment.

However, IBM does not supply these for the cross-platform hosts.

If your cross-platform host supports NFS or SMB network file access, these files can be made available to the cross-platform host. For more information about how to configure and run the NFS server on z/OS, see the IBM document *NFS Customization and Operation*. For information about running an SMB server on your z/OS host, see the IBM document *Distributed File Service SMB Administration*.

Input Files

DASM can process either ASCII or EBCDIC input files. The assembler supports automatic searches for MACRO and COPY files with various options to describe the search location for these.

EBCDIC input files are read 80 bytes at a time, with no new-line markers. ASCII input files are read a line at a time. On z/OS, the assumption is that the input is an FB-80 EBCDIC file.

The sequence number information in columns 73-80 is ignored.

If the input file name specified on the command line is a hyphen ("-"), no file will be opened and standard input will be used instead. In this fashion you can use **DCC** or **DCXX** with the "-o-" option piped to **DASM** to avoid generating a temporary file between these two steps.

ASCII/EBCDIC translation

On ASCII-based hosts **DASM** can process either ASCII or EBCDIC input files. **DASM** will read the first 80 bytes of an input file, looking for an ASCII new-line or other ASCII control character. If none is found, the file is assumed to be an EBCDIC input file and no translation is performed. All of the statements in the file must be either EBCDIC or ASCII.

If **DASM** determines the file is ASCII, it reads each source line up to the ASCII new-line character, padding the line to 80 characters with blanks. The input line is then translated internally to EBCDIC for continued processing.

By default, **DASM** uses a variant of the IBM1047 code page. The traditional definition of this code page maps EBCDIC `X'15'` to ASCII `X'0A'` and EBCDIC `X'25'` to ASCII `X'85'`. The official IBM1047 definition has this reversed. Some tools available from different vendors use the traditional IBM1047 mapping, and some use the official mapping. However, `DASM` adopts the traditional mapping

The ASCII/EBCDIC translation tables can be adjusted from the **DASM** defaults using the *–tr* option.

Output Files

The assembler generates a listing output file unless options are set to disable it. The location of the listing file is platform specific, and can be redirected via various options.

On z/OS, the assembler will generate an object file by default. On cross-platform hosts, the assembler will generate an object file only if the appropriate option is present on the command line.

The assembler can also generate other files as described in the various option descriptions.

The assembler will also write messages to the *stderr* file stream.

Running DASM:

The **DASM** command is used to assemble source programs and generate object code.

z/OS:

In the z/OS environment, the assembler is executed by invoking the `DASM` member of the Systems/ASM installation PDS. The options are specified in the `PARM` statement. Each **DASM** option is separated by a comma, and preceded with a dash.

On z/OS, the assembler begins with certain option defaults which can be modified via the `PARM` statement. An option which begins with the commercial at-sign (`@`), specifies a DD from which to read other options.

By default, the input source is assumed to be the `SYSIN` DD, the assembled object is written to the `SYSLIN` DD and the listing is generated on the `SYSPRINT` DD. There is a default -L option supplied which specifies a template of the form `//DDN:SYSLIB(&M)`. This template causes **DASM** to search the `SYSLIB` DD for MACRO and COPY members.

Furthermore, the assembler may require a definition of the `SYSUT1` DD for temporary storage.

For example, the following JCL will assemble the source on the `SYSIN` DD, generating the object `MYOBJ` in the `MY.OBJS` PDS, printing the listing on the `SYSPRINT` DD:

```
//DASM JOB
//DASM  EXEC PGM=DASM
//STEPLIB DD DSN=DIGNUS.LOAD,DISP=SHR
//SYSLIN  DD DSN=MY.OBJS(MYOBJ),DISP=OLD
//SYSLIB  DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD DSN=&&SYSUT1,
//           SPACE=(4096,(120,120),,,ROUND),
//           UNIT=VIO,DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//SYSIN    DD *
     <assembler source>
```

Windows:

When using the Windows operating systems, the assembler is named **dasm** and may be found in the installation directory. The command line is

```
dasm [options] input-file.asm
```

Options, if any, are preceded with a dash, `-`.

Unless otherwise specified, the assembler does not generate an object file. Use the *–o* option to specify the name of the generated object file.

The Windows version of **DASM** supports the *–@filename* option. *–@filename* causes the compiler to read the file filename and insert its contents into the command line. This provides a mechanism for supporting arbitrarily long command line parameter lists.

In a Windows environment, **DASM** examines the `DASMINI` environment variable. The environment variable may contain a list of semi-colon delimited files that will be treated exactly as if they had be specified with the *–@filename* option.

Note that on Windows, the ampersand (`&`) character has special meaning. Double quotes will need to be employed to prevent the Windows `COMMAND.COM` processor from interpreting ampersand characters.

UNIX:

In the UNIX environment, the assembler is named **dasm**, and can be found in the installation directory. The command line is

```
dasm [options] input-file.asm
```

Options, if any, are preceded with a dash, `-`.

Unless otherwise specified, the assembler does not generate an object file. Use the *–o* option to specify the name of the generated object file.

In a Windows environment, **DASM** examines the `DASMINI` environment variable. The environment variable may contain a list of semi-colon delimited files that will be treated exactly as if they had be specified with the *–@filename* option.

The UNIX version of **DASM** supports the *–@filename* option. *–@filename* causes the compiler to read the file filename and insert its contents into the command line. This provides a mechanism for supporting arbitrarily long command line parameter lists.

Note that on Unix, the ampersand (`&`) character has special meaning. Quotation marks will need to be employed to prevent the command interpreter from intecepting and interpreting ampersand characters.

Macro Library Searching:

The assembler searches for MACRO and COPY members based on the *–L* and *–macext* options. These options provide two mechanisms for searching for MACRO and COPY files.

The *–L* option can be used to either specify a simple location to search for members, or to specify a macro search template which will be expanded with the macro name and other substitution values. If the *–L* specification does not contain any substitution characters, then it is assumed to be a simple location.

If the *–L* option specifies a simple location then the *–macext* option can be used used to provide a default macro name extension.

If a *–L* specification does not contain any substitution characters, it is a simple location. In this situation, the assembler creates a file name by first lower-casing the MACRO/COPY name and appending the extension specified in the *–macext* option. It then uses the value from the *–L* option to form a complete file name and attempts to open that file to satisfy the MACRO/COPY request. If that file doesn't exist, the assembler then uses the upper-case version of the MACRO/COPY name, with the *–macext* specified extension, combined with the location specified in the *–L* option. Note that if no *–macext* option is specified, the assembler uses a default extension of "`mac`".

If the *–L* specification does contain subsitution characters, then the specification is considered to be a search template.

When a search template is specified, the *–macext* value is not appended to generate the file name.

When the *–L* specification is a search template, the assembler performs the replacements for the substitution characters as described below and tries to open the resulting file.

Each *–L* specification can contain multiple locations or templates, separated by the separation character; or the *–L* option can appear multiple times on the command line. The separation character is a semicolon (`;`). On some UNIX platforms, the colon (`:`) can also be used as the separation character, for backward compatibility with previous releases.

Searches for MACRO/COPY files proceed through the specifications of locations and templates in the *–L* options in the specified order.

When using *–L* to specify a search template, the following substitution characters may be used:

`&d`	Source drive on Windows, empty on other hosts.
`&D`	Source file directory. On Windows, this includes the drive letter.

`&e`	Source file extension.
`&f`	Source file, without any directory/path components and without any extension.
`&p`	Path to source file. On Windows, this does not include the drive letter.
`&M`	Upper case version of the name provided in the MACRO/COPY statement.
`&m`	Lower case version of the name provided in the MACRO/COPY statement.
`&x`	Directory where the **DASM** executable was found.
`&&`	The `&` character.

For example, if **DASM** was running on a UNIX platform, and the macros were presented in the `./mymacros` and `/usr/local/sysmacros` directories. And, the macros in these directories had no file name extension. Furthermore, we would like **DASM** to examine the `./mymacros` directory first, before looking in `/usr/local/sysmacros`, the following command line could be used:

```
dasm '-L./mymacros/&M' '-L/usr/local/sysmacros/&M' ...
```

(Note the use of the single quote character to prevent the UNIX shell from interpreting the `&`-character.)

If we assume that *NAME* is the name of the MACRO or COPY file of interest, then these options indicate that **DASM** will first take the *NAME* convert it to upper-case (because of the `&M` substituation) and look for the file `./mymacros/`*NAME* followed by `/usr/local/sysmacros/`*NAME*.

ZIP file support

On cross-platform hosts, **DASM** supports searching ZIP archives for MACRO or COPY members. A search path specification which contains parenthesis will be examined to determine if it is a ZIP archive. If so, members of the archive will be automatically extracted from the archive and included in the assembly.

For example, on a UNIX host, the search specification:

```
'-L./maclib.zip(&M)'
```

will search for macro and copy members from the ZIP library named `maclib.zip`.

Description of Options

The options available to **DASM** are summarized in the following sections.

General Options

These options are general in nature. They are summarized in the table below.

–fdate=*[[[[[cc]yy]mm]dd]HH]*MM*[.ss]]*	specify assembly time and date
–flisting=*file*	specify the name of the listing file
–flisting_style=*val*	specify listing format
–fhtml_suffix=*val*	specify HTML listing suffix
–? –h –help	display help on **DASM** options
–hl –helplib	display help for **DASM** library specifications
–o *file*	place any generated object in the file named *file*
–q –quiet	execute **DASM** in "quiet" mode
–@*filename*	specify extra file for parameters
–fmrc –fnomrc	enable/disable mainframe-style return-codes
–fmaxrc=*n*	don't create an object file if the RC is greater than *n*
–v	print version information and exit

The –fdate=*[[[[[cc]yy]mm]dd]HH]*MM*[.ss]]* option (specify assembly time and date)

The *–fdate* option sets the "assembly time and date" for the current assembly. The assembler uses this value as the time and date in the assembly listings, as the value of the `SYSCLOCK` macro variable, as the time value for any `AREAD` statements and for the timestamp on an assembler-generated `END` card in the object deck.

All other time values, e.g. the elapsed time for the assembly, assembler start/stop time in `ADATA` information, are unaffected by this option.

This option can be useful in reassembling a source with a given timestamp, so any differences from a previous assemble can be detected.

The value specified is taken as overriding to the current local time, only the `MM` portion has to be specified, the remaining values are optional. If a value isn't specified, the current local time value is used.

The values are:

cc	Century (either 19 or 20) prepended to the abbreviated year.
yy	Year in abbreviated form (e.g. 89 for 1989, 06 for 2006).
mm	Numeric month, a number from 1 to 12.
dd	Day, a number from 1 to 31.
HH	Hour, a number from 0 to 23.
MM	Minutes, a number from 0 to 59.
ss	Seconds, a number from 0 to 61 (59 plus a maximum of two leap seconds).

All values are optional, except `MM`.

The –flisting=*file* option (specify the name of the listing file)

The *–flisting=file* option instructs the assembler to use the given file name for the generated listing file. When *–flisting=file* isn't specified, the assembler uses the default name for producing the listing. On cross-platform hosts, the default listing file name is based on the input file name, with a suffix of `.lst`. On z/OS, the listing file name is the `SYSPRINT` DD.

The –flisting_style=*val* option (specify listing format)

The *–flisting_style=val* option controls the format of the listing. *val* may be *none*, *txt*, *html*, or *both*. The default is as if *–flisting_style=txt* were specified, the listing contains mono-spaced plain text with carriage control codes. If *–flisting_style=html* is specified, then an HTML listing file is created. If *–flisting_style=both* is specified then both the HTML and text listings are produced simultaneously.

The HTML listing allows you to view the listing in your web browser. In addition to some aesthetic improvements, the HTML listing adds hyperlinks in the cross-reference sections so you can easily find the line that provides a definition or reference.

When generating an HTML listing, the same filename is used as for a text listing, except that the suffix is replaced with `.html`. Use *–fhtml_suffix=val* to specify an alternative filename suffix. If **DASM** is running on MVS or CMS, then the HTML listing is unconditionally written to the `SYSHTML` DD.

The HTML listing has virtual page breaks every 30 lines in the `Statement Listing` section. These page breaks provide the current `TITLE` and `Active Usings` references. If *–lc* is specified, then that value will be used instead.

The –fhtml_suffix=*val* option (specify HTML listing suffix)

To provide an alternative to `.html` for the suffix of the HTML listing filename, use the *–fhtml_suffix=val* option. For example, *–fhtml_suffix=.htm* would use the shorter `.htm` suffix for compatibility with legacy utilities that cannot use suffixes longer than 3 characters.

The –help option (display help)

The *–help*, *–h* and *–?* options will cause **DASM** to produce a summary of the available options.

The –helplib option (display help for DASM library specifications)

The *–helplib* and *–hl* options will cause **DASM** to produce a summary of the various library search options.

The –o *file* option (specify the name of the generated output file)

The *–ofile* option specifies the name of the assembler output file. On cross-platform hosts, if *–ofile* isn't specified, no output is written. On z/OS, the output is written to the `SYSLIN` DD by default.

Substitution characters can be specified in the *file* name. The substitution characters in a *–o* option expand as follows:

`&d`	Source drive on Windows, empty on other hosts.
`&D`	Source file directory. On Windows, this does include the drive letter.
`&e`	Source file extension.
`&f`	Source file, without any directory/path components and without any extension.
`&p`	Path to source file. On Windows, this does not include the drive letter.
`&x`	Directory where the **DASM** executable was found.
`&&`	The `&` character.

The –quiet option (execute DASM in "quiet" mode)

The *–quiet* option suppresses generation of the **DASM** banner and summary messages.

The –@ *file* option (specify extra file for parameters)

The *–@file* option specifies the name of another input file which contains more **DASM** options. *file* contains one option per line.

Substitution characters can be specified in the *file* name. The substitution characters in a *–@* option expand as follows:

`&d`	Source drive on Windows, empty on other hosts.
`&D`	Source file directory. On Windows, this does include the drive letter.
`&e`	Source file extension.
`&f`	Source file, without any directory/path components and without any extension.
`&p`	Path to source file. On Windows, this does not include the drive letter.
`&x`	Directory where the **DASM** executable was found.
`&&`	The `&` character.

The contents of the *file* are interpreted as if they appeared on the **DASM** command line in the location of the *–@* option.

The –fmrc and –fnomrc options (enable/disable mainframe-style return-codes)

The *–fmrc* option causes **DASM** to use mainframe-style return codes, and is the default when **DASM** is running on a mainframe host. Mainframe style return codes are 0 for success, 4 for warnings, 8 for errors, and 12 for catastrophic errors. The *–fnomrc* option causes **DASM** to use Unix-style return codes, and is the default on non-mainframe hosts. Unix style return codes are 0 for success, or 1 for any warning or errors at all.

The –fmaxrc=*n* option (don't create an object file if the RC is greater than *n*)

The *–fmaxrc=n* option causes **DASM** to create an object file only if the return code is less than or equal to *n*. By default **DASM** will create an object file even if there are errors.

The –v option (print version information)

The *–v* option causes **DASM** to print the version information on the STDERR stream and exit with a return code of 0.

Input Options

The following options affect processing of the input assembler source.

–flonglines	Allow input lines longer than 80 characters
–fnolonglines	Diagnose input lines longer than 80 characters
–tc *column*	define the tab expansion column
–tr *XX=YY*[,*XX=YY*...]	alter the ASCII/EBCDIC translation table

The –flonglines option (allow input lines longer than 80 characters)

Normally, when the assembler encounters a source input line longer than 80 characters it generates diagnostic warning DASM900W, and truncates the input line to 80 columns.

If the *–flonglines* options is enabled, the assembler will continue to truncate the input line to 80 columns, but the warning will be suppressed.

Thus, when *–flonglines* is specified, the assembly source can contain text past column 80 that will be silently ignored.

The –fnolonglines option (Diagnose input lines longer than 80 characters)

The *–fnolonglines* option causes the assembler to produce diagnostic DASM900W indicating an input source line was longer than 80 characters. The input source line will be truncated to 80 characters and assembly will continue.

This is the default.

The *–flonglines* option can be used to suppress message DASM900W.

The –tc *column* option (define tab expansion amounts)

The *–tc column* option specifies the column number multiple to use for expanding tab characters in the input assembler source. The default value for *column* is `0`.

By default, the tab character is not expanded; because the tab character can appear in comments and define constant values, and cannot be expanded to spaces in those situations.

Previous versions of DASM would default this to 8, to get the previous that behavior specify an *–tc 8* option.

The –tr *XX*=*YY*[,*XX*=*YY*...] option (alter the ASCII/EBCDIC translation table)

When ASCII input is discovered, **DASM** translates the ASCII input to EBCDIC for internal processing. The translation table **DASM** employs is described in an appendix in this document.

In some situations, it might be useful to alter the translation table from the **DASM** default.

The *–tr* option specifies an ASCII hexadecimal value, *XX*, that is translated to an EBCDIC hexadecimal value, *YY*. Multiple *XX*=*YY* values can be specified at a time, separated by commas. Also the *–tr* option can be specified as many times as needed.

If the *YY* EBCDIC value is not specified, it indicates that the translation should return to the **DASM** default.

For example, the command line option:

```
-tr 25=0A,4A=5B
```

specifies that the ASCII character hex value 25 (decimal value 37) should be translated to the EBCDIC hex value 0A (decimal value 10), and the ASCII hex value 4A (decimal value 74) would be translated to the EBCDIC hex value 5B (decimal value 91.)

To adjust the **DASM** code page table from the IBM1047 variant to IBM037, these options would be used:

```
-tr 5E=B0,AC=5F,5B=BA,5D=BB,DD=AD,A8=BD
```

To adjust **DASM** to use the official IBM1047 mapping instead of the traditional one, these options would be used:

```
-tr 0A=25,85=15
```

Any *–tr* values specified will appear in the Option Summary section of the listing.

Assembly Control Options

The following options relate to controlling various aspects of the assembly process.

–sectalgn=*VALUE*	define section alignment
–size=*VALUE*	define the amount of memory used for temporary work files
–sysparm=*VALUE*	define the `&SYSPARM` variable to have the value *VALUE*
–a / –xa	enable/disable non-strict alignment checks
–r / –xr	enable/disable re-entrancy checks
–pestop / –xpestop	enable/disable halt on *PROCESS and command-line errors
–profile=*filename* / –xprofile	include *filename* as if it appeared in a COPY statement
–ra2 / –xra2	enable/disable 2-byte relocatable address constant checks
–tc / –xtc	enable/disable checks for immediate and register operands
–tc *val*	*val* specifies register and/or immediate operand type checks
–f *integer*	set the FLAG value for controlling diagnostics
–f ALIGN / –f NOALIGN	set the FLAG value for strict alignment checks
–f CONT / –f NOCONT	set the FLAG value for continuation checks
–f EXLITW / –f NOEXLITW	issue warning message #016 when a literal is the the target of an EX instruction
–f IMPLEN / –f NOIMPLEN	issue warming message DASM169I when an implied length is used in an SS-type instruction
–f PAGE0 / –f NOPAGE0	set the FLAG value for enabling/disabling message DASM309W
–f USING0 / –f NOUSING0	set the FLAG value for enabling/disabling message DASM306W
–f PUSH / –f NOPUSH	set the FLAG value for enabling/disabling non-empty PUSH stack checks
–f RECORD / –f NORECORD	set the FLAG value for enabling/disabling message DASM435I

–f SUBSTR / –f NOSUBSTR	set the FLAG value for enabling/disabling message DASM094I
–ccase / –xccase	enable/disable case compatibility with ASMH
–mcase / –xmcase	enable/disable macro case compatibility with ASMH
–csysl / –xcsysl	enable/disable SYSLIST compatibility with ASMH
–clit / –xclit	enable/disable literal type compatibility with ASMH
–csyspath / –xcsyspath	enable/disable &SYSxxx paths compatible with HLASM on USS
–xsd / –xxsd	enable/disable XSD-style objects
–goff / –xgoff	enable/disable GOFF-style objects
–goffadata / –xgoffadata	enable embedded ADATA information in GOFF objects
–thread / –xthread	enable/disable CSECT threading
–fenhancedequ	enable enhanced EQU evaluation
–batch / –xbatch	batch source processing
–idr *VALUE*	define the IDR information on END cards
–fdupalias	allow duplicate ALIAS values
–fvselibr	recognize /+ as EOF
–fsuprwarn=*list* / –fnosuprwarn=*list*	suppress or don't suppress particular warning messages
–fasciiout / –fno_asciiout	enable/disable ASCII character constants
–flinux	generate Linux/390 ELF output
–flinux64	generate z/Linux ELF output
–g / –xg	enable/disable Linux ELF STABS debugging output
–fdwarf=*file*	enable output of DWARF side file
–fmapat –fnomapat	enable/disable mapping '@' to '_' in external symbol names

The –sectalgn=*VALUE* option (define section alignment)

The *–sectalgn=VALUE* option specifies the minimum alignment for sections and literals.

VALUE is an power-of-2 integer in the range from `8` to `4096`.

If a *VALUE* other than 8 is specified, the *–goff* option must also be specified. `GOFF` objects have provisions for specifying other section alignment to the linker.

The –size=*VALUE* option (define memory for in-core work files)

The *–size=VALUE* option defines the amount of memory the assembler can allocate for in-memory work files. *VALUE* can be either `0` or `MAX`.

The default *VALUE* is `MAX` which indicates the assembler should use all available memory for temporary work files. This provides the best assembly-time performance.

A value of `0` indicates that the assembler should use no memory for temporary work files. Instead, the work file will be on disk. Depending on file system performance, this can slow down the assembly process.

The value of `0` can be specified if the assembler runs out of memory during large assembly runs.

The –sysparm=*VALUE* option (define a SYSPARM value)

The *–sysparm=VALUE* option defines the value of the `&SYSPARM` variable.

The –a and –xa options (enable/disable non-strict alignment checks)

DASM will perform various alignment checks on referenced data and literals. If alignment checks are enabled, a mis-aligned access will produce an assembler warning. The default is to perform alignment checks.

These alignment checks apply only to non-strict alignment requirements, and generate message DASMA033. Strict alignment checks, such as branch alignments and operands that require alignment for operation are controlled by the *–f ALIGN* and *–f NOALIGN* option.

Note that if *–xa* is enabled, the assembler will only align DC, DS, DXD and CXD values on correct boundaries if their duplication factor is 0. If *–a* is enabled, DC, DS, DXD and CXD values will be aligned on correct boundaries.

The –r and –xr options (enable/disable re-entrancy checks)

DASM will perform various checks on the generated code to assure it can participate in a RENT module. If re-entrancy checks are enabled, a reference which alters a CSECT will produce an assembler warning. Re-entrancy checks are enabled by default.

Note that some non-re-enterable code may not be identified with a message as the assembler cannot exhaustively check program logic code.

The –pestop and –xpestop options (enable/disable halt on *PROCESS and command-line errors)

If the *–pestop* option is enabled, the assembler will terminate when an error is detected in the command-line options, or in `*PROCESS` statements.

The default is *–pestop*.

The –profile=*filename* and –xprofile options (include *filename* as if it appeared in a COPY statement)

The *–profile=filename* option causes **DASM** to insert the file named *filename* as if it appeared on a `COPY` statement at the start of the program.

The text from *filename* will appear at the start of the program, after any `*PROCESS` statements.

The *–xprofile* option disables this feature, and is the default.

The *–profile* and *–xprofile* options may be abbreviated as *–prof* and *–xprof*.

The `PROFILE` option can also be used in `*PROCESS` statements.

The –ra2 and –xra2 options (enable/disable 2-byte relocatable address constant checks)

DASM will generate message number DASM066W if a Y-type or 2-byte A-type relocatable address constant is discovered. To disable this message, use *–xra2*.

The default value is *–ra2*.

The –tc and –xtc options (enable/disable range checks for immediate operands)

DASM will generate warning message DASM320W for signed immediate operands which are outside the appropriate range. *–tc* is analagous to the HLASM `TYPECHECK(MAGNITUDE,REGISTER)` option.

The default value is *–tc*.

The *–tc* option will optionally accept a comma, or colon separated value specifying `magnitude` and/or `register`. `magnitude` may be abbreviated as `mag` and `register` may be abreviated as `reg`.

For example:

```
-tc reg:mag
```

specifies both the `REGISTER` and `MAGNITUDE` sub-options of `TYPECHECK`.

If the *–tc* option is specified alone, it is equivalent to specifying both `REGISTER` and `MAGNITUDE`.

The –f ALIGN and –f NOALIGN options (set the FLAG value for strict alignment checks)

The *–f ALIGN* option controls the generation of diagnostic messages that check instruction alignment, branch alignment and strict operand alignment.

If *–f NOALIGN* is specified, the assembler will not generate diagnostic messages DASM033I, DASM212W or DASM213W.

If *–f ALIGN* is specified, the assembler will generate diagnostic messages DASM212W and DASM213W. If the *–a* option is also enabled, the assembler will generate message DASM033I.

Diagnostic message DASM212W refers to branch alignment requirements, message DASM213W applies when the instruction operand requires alignment for proper operation.

The –f CONT and –f NOCONT options (set the FLAG value for continuation checks)

The *–f CONT* option (enabled by default), causes the assembler to check for malformed continuation lines and issue DASM430W, DASM431W, DASM432W and DASM433W.

Enabling *–f NOCONT* disables these messages.

The –f IMPLEN and –f NOIMPLEN options (issue warming message DASM169I when an implied length is used in an SS-type instruction)

The *–f IMPLEN* option enables the generation of information message DASM169I, *–f NOIMPLEN* disables it.

When *–f IMPLEN* is enabled, **DASM** generates message DASM169I when the operand with an implied length in an SS-type instruction is discovered.

When *–f NOIMPLEN* is enabled, message DASM169I will not be generated,

By default, *–f NOIMPLEN* is enabled.

The –f PAGE0 and –f NOPAGE0 options (set the FLAG value for enabling/disabling message DASM309W)

The *–f PAGE0* option controls checking for missing base registers in memory addresses.

If *–f PAGE0* is specified the assembler flags any machine instruction operand that is a memory address reference but doesn't specify a base register with the diagnostic message DASM309W.

Note that this is only for instructions that actually reference the memory, instructions that only compute the address (e.g. LOAD ADDRESS) have no check.

Furthermore, if a base register is not specified, but an index register is specified, the warning isn't generated.

The *–f NOPAGE0* option will disable the warning.

The –f USING0 and –f NOUSING0 options (set the FLAG value for enabling/disabling message DASM306W)

If *–f USING0* is specified (the default), then **DASM** will generate the diagnostic `DASM306W` for any `USING` that conflicts with the default `USING 0,0`, subject to the bits specified in the *–uwarn n* option.

The –f PUSH and –f NOPUSH options (set the FLAG value for enabling/disabling non-empty PUSH stack checks)

The *–f PUSH* option controls the generation of diagnostic messages that check for non-empty PUSH stacks at the end of the assembly.

If *–f PUSH* is enabled, a DASM138W message can be generate if the PRINT, USING or ACONTROL push stacks are non-empty at the end of the assembly source.

Use *–f NOPUSH* to disable this message.

By default, *–f PUSH* is enabled.

The –f RECORD and –f NORECORD options (set the FLAG value for enabling/disabling message DASM435I)

The *–f RECORD* option enables the generation of information message DASM435I, *–f NORECORD* disables it.

When *–f RECORD* is enabled, **DASM** generates message DASM435I after the last error messages for any statement. This message contains the line number, and file name information for the statement in error.

Also, when *–f RECORD* is enabled, the "Diagnostic Cross Reference and Assembler Summary" section of the assembler listing will include the file name and line number for each statement in error.

When *–f NORECORD* is enabled, message DASM435I will not be generated, and the "Diagnostic Cross Reference and Assembler Summary" section will only contain the statement number of any diagnosed lines.

By default, *–f RECORD* is enabled.

The –f SUBSTR and –f NOSUBSTR options (set the FLAG value for enabling/disabling message DASM094I)

The *–f SUBSTR* option enables the generate of information message DASM094I, *–f NOSUBSTR* disables it.

When *–f SUBSTR* is enabled, **DASM** checks every string subscript option to ensure the second subscript specifies a length that is not longer then the given string value. If the length is longer, the informational message DASM094I will be generated.

When *–f NOSUBSTR* is enabled, message DASM094I will not be generated.

The default is *–f NOSUBSTR*.

The –f *integer* option (set the FLAG value for controlling diagnostics)

The *–f integer* option specifies that messages with *integer* or higher severity code will be generated. Messages with a severity code lower than *integer* are suppressed. The default value of code is `0`.

The –ccase and –xccase options (enable/disable case compatibility with ASMH)

The *–ccase* and *–xccase* options enable and disable the ASMH case compatibility feature.

The *–ccase* option causes the assembler to maintain compatibility with the IBM ASMH assembler. Assembler pseudo-ops, instructions and other language elements are restricted to upper case if they were so restricted with IBM's earlier ASMH assembler.

The default is *–xccase*.

The –mcase and –xmcase options (enable/disable macro case compatibility with ASMH)

The *–mcase* and *–xmcase* options enable and disable the ASMH macro case compatibility feature.

The *–mcase* option causes the assembler to maintain uppercase compatibility with the IBM ASMH assembler for unquoted macro operands. If *–mcase* is enabled, the assembler will convert lower case alphabetic characters (characters `a` through `z` in unquoted macro operands to uppercase.

The default is *–xmcase*.

The –csysl and –xcsysl options (enable/disable SYSLIST compatibility with ASMH)

The *–csysl* option causes the assembler to treat sublists in `SETC` symbols in a manner compatible with the earlier IBM ASMH assembler. When *–csysl* is enabled, `SETC` symbols that are assigned parenthesized sublists are treated as character strings, not sublists, when passed as an operand of a macro instruction.

The default is *–xcsysl*.

The –clit and –xclit options (enable/disable literal type compatibility with ASMH)

The *–clit* and *–xclit* options enable and disable the ASMH literal type compatibility feature.

The *–clit* option causes the assembler to maintain compatibility with the IBM ASMH assembler. ASMH always considers the type attribute of a literal to be `U` when

it appears in an expression such as `T'=X'11'`. When compatibility mode is not enabled, the actual type of the literal is used if the literal is currently pending from any preceding code without an intervening LTORG.

The default is *–xclit*.

The –csyspath and –xcsyspath options (enable/disable &SYSxxx paths compatible with HLASM on USS)

DASM provides several pre-defined set-symbols to describe the files used for `SYSIN`, `SYSLIB`, `SYSLIN`, `SYSPRINT`, `SYSPUNCH`, `SYSADATA`, and `SYSTERM`. For example, `&SYSIN_DSN`, `&SYSIN_MEMBER`, and `&SYSIN_VOLUME` provide the location of the primary assembly source.

By default, DASM puts the drive letter (Windows only) into the `&SYSxxx_VOLUME` set-symbol. The complete absolute directory path is uppercased and then placed in `&SYSxxx_DSN`. The uppercased filename (with no extension) is placed in `&SYSxxx_MEMBER`.

However, HLASM on USS places the full un-modified path and filename in `&SYSxxx_DSN` and leaves the other set-symbols empty. If *–csyspath* is specified, then **DASM** will do the same.

The –xsd and –xxsd options (enable/disable XSD-format objects)

DASM can generate ESD-style, GOFF-style or XSD-style object deck formats. If using label names longer than 8 characters for either section names or names which are the target of ENTRY statements, the GOFF or XSD-style format should be used. This format allows for names of any length in the generated object deck, and is compatible with the Systems/C pre-linker **PLINK** and IBM binder.

The default is to use XSD-style object decks.

More information regarding XSD-style object decks can be found in the IBM manual entitled *DFSMS/MVS Program Management*.

The –goff and –xgoff options (enable/disable GOFF-format objects)

DASM can generate ESD-style, GOFF-style or XSD-style object deck formats. If using label names longer than 8 characters for either section names or names which are the target of ENTRY statements, the GOFF or XSD-style format should be used. This format allows for names of any length in the generated object deck, and is compatible with the Systems/C pre-linker **PLINK** and IBM binder.

Note that the **DASM** default is to produce XSD-style object decks.

More information regarding GOFF-style object decks can be found in the IBM manual entitled *DFSMS/MVS Program Management*.

The –goffadata and –xgoffadata options (enable embedded ADATA information in GOFF objects)

As well as producing GOFF-style object decks, **DASM** can include ADATA debugging information. The GOFF definition allows for this information to be embedded within the generated object deck, or separately. The *–goffadata* option causes the assembler to embed this information within the object deck.

Note that the *–A=file* option causes the assembler to produce this information as a separate file.

The –batch and –xbatch options (enable/disable batch source processing)

The *–batch* option indicates that the input file may contain multiple assembler sources. The separate assembler sources are distinguished with the `END` statement. The *–xbatch* option indicates there is only one assembler source in the input file.

The default mode is *–batch*.

The –thread and –xthread options (enable/disable CSECT threading)

When *–thread* is specified, **DASM** assigns the starting offset for a new CSECT to the ending address of the previous CSECT. When *–xthread* is specified, **DASM** assigns all CSECTs to begin at 0. The default is *–thread*.

The –fenhancedequ option (enable enhanced EQU evaluation)

When *–fenhancedequ* is specified, **DASM** allows EQUs to remain unresolved for longer, allowing more complex dependencies between EQUs. By default **DASM** is very strict and only accepts EQUs which can be resolved in the first pass for compatibility with HLASM.

The –idr option (define the IDR string generated on END cards)

The *–idr* option defines the identification information (IDR) generated on END cards produced by the assembler. Some software examines IDR information to

determine the component that produced the object. This option can be useful when the generated object should appear to have been produced by a different product.

By default, the IDR appears as

```
DASMvv   rrll
```

where *vv* is the **DASM** major version number, *rr* is the release number and *ll* is the modification level.

The –fdupalias option (allow duplicate ALIAS values)

The *–fdupalias* option extends the assembler to support more than one symbol aliases to the same value.

For example, the following two ALIAS statements would normally be flagged as an error:

```
NAME1 ALIAS C'value'
NAME2 ALIAS C'value'
```

because the two symbols `NAME1` and `NAME2` are aliases to the same character constant, `C'value'`. With the *–fdupalias* option, the assembler will not issue a warning or message of any kind and will honor the ALIAS statements.

The *–fdupalias* option is typically used when assembling source produced by the Systems/C and Systems/C++ compilers.

The –fvselibr option (recognize /+ as EOF)

On VSE, when placing members into VSE LIBRARY library, the special characters `/+` can mark the end-of-file when the member is retrieved.

That is, if a record begins with the two character `/+` the characters and record that follow are ignored.

This is sometimes used to place comments into the member, without being considered part of the member data.

To facility the use of such files on cross-platform hosts, the *–fvselibr* option causes **DASM** to consider a source line that begins with `/+` as an end-of-file marker. When the marker is encountered at the beginning of an input line, **DASM** will discard the line and any further data contained in the file.

The –fsuprwarn=*list* and –fnosuprwarn=*list* options (suppress or don't suppress particular warning messages)

The *–fsuprwarn* and *–fnosuprwarn* options accept a colon-separated list of integers specifying particular warning message numbers to suppress (or not suppress.)

At least one integer value must be specified in the colon-separated *list*.

A message is generated if the given message number does not exist, or the message severity is not 4 or less.

The –fasciiout and –fno_asciiout options (enable/disable ASCII character constants)

When *–fasciiout* is specified, character constants such as

```
STR  DC C'HELLO'
```

generate bytes encoded in the ASCII character set. If *–fno_asciiout* is specified then EBCDIC character constants are generated. *–fno_asciiout* is the default, unless the *–flinux* option is specified.

The –flinux option (generate Linux/390 ELF output)

The *–flinux* option instructs **DASM** to generate a Linux/390 compatible ELF (`.o`) file from the HLASM compatible source rather than a traditional object deck. See the "Linux ELF Output" section for more information. *–flinux* implies *–fasciiout*.

The –flinux64 option (generate z/Linux ELF output)

The *–flinux64* option instructs **DASM** to generate a z/Linux compatible ELF (`.o`) file from the HLASM compatible source rather than a traditional object deck. 64-bit z/Linux objects are incompatible with 32-bit Linux/390 objects, so care must be taken in chosing the correct one of *–flinux* or *–flinux64*. See the "Linux ELF Output" section for more information. *–flinux64* implies *–fasciiout*.

The –g and –xg options (enable/disable Linux ELF STABS debugging output)

The *–g* option causes any generated Linux ELF object files to contain "STABS" debugging output. When this debugging output is present, gdb supports assembly

source level debugging on the generated objects. The debugger is given access to the line numbers of the original input file and also is presented with `struct` definitions for the **DSECT**s. The *–xg* option disables debugging output and is the default.

The –fdwarf=*file* option (enable output of DWARF side file)

If you have instructed **DCC** or **DCXX** to generate DWARF debugging information, then *–fdwarf=file* specifies the output filename for the DWARF information. The compiler emits a `*PROCESS DWARF('filename')` line, which provides the default filename for DWARF information. You only need to specify *–fdwarf=filename* if you wish to override the value the compiler selected.

The –fmapat and –fnomapat options (enable/disable mapping '@' to '_' in external symbol names)

If *–fmapat* is specified then any at signs ('`@`') in external symbol names (after `ALIAS` mapping) will be replaced with underscores ('_'). This option is especially useful for Linux (ELF) mode, where at signs are not valid in symbol names.

Library options

–macext *extension*	specify the extension to use for MACRO/COPY file names
–L *location*	specify the location to search for MACRO/COPY members
–V *env*	name an environment variable that specifies search locations for macro/copy members
–libexec *cmd*	provide a program to execute when the assembler cannot find a macro/copy member
–libncase	use case-insensitive filename search

The –macext option (specify the extension to use for MACRO/COPY file names)

The *–macext* option specifies the extension to use when searching for MACRO/COPY file names with a *–L* specification that does not contain a substitution character. The default extension used is "`mac`". To indicate that no extension should be used, use a single period as the value in the *–macext* option.

For example, if the command line include `-L/usr/maclib` and the *–macext* option was set to `MAC`, and the assembler was searching for the macro `MYMAC`, the assembler would look for

```
/usr/maclib/mymac.MAC
```

followed by

```
/usr/maclib/MYMAC.MAC
```

The –L option (specify the location to search for MACRO/COPY members)

The *–L* option specifies a location to search for MACRO and COPY members. More than one *–L* option may be specified. If the value specified in a *–L* option does not use the `&M`, `&m` or `&D` substitution sequence, then the member name, followed by the macro extension will be appended to the value specified in the *–L* option to locate the MACRO or COPY member. The default macro extension may be modified with the *–macext* option.

The substitution characters in a *–L* option expand as follows:

`&d`	Source drive on Windows, empty on other hosts.
`&D`	Source file directory. On Windows, this does include the drive letter.
`&e`	Source file extension.
`&f`	Source file, without any directory/path components and without any extension.
`&p`	Path to source file. On Windows, this does not include the drive letter.
`&M`	Upper case version of the name provided in the MACRO/COPY statement.
`&m`	Lower case version of the name provided in the MACRO/COPY statement.
`&x`	Directory where the **DASM** executable was found.
`&&`	The `&` character.

When no substitution character is specified, the assembler first tries the lower-case version of the MACRO/COPY name (followed by the macro extension), then the upper-case version of the name.

For example, if the `-L /usr/maclib` option was specified, and the assembler discovered an invocation of the MYMAC macro, and the default macro extension had not been overridden, the assembler would look for

```
/usr/maclib/mymac.mac
```

and then

```
/usr/maclib/MYMAC.mac
```

If the `-L..\mymacros\&M` option had been specified, the assembler would look for the file

```
..\mymacros\MYMAC
```

The specified *–L* options are searched in the order they appear in the **DASM** options.

Note that on cross-platform hosts, if the *–L* option specifies a pattern which contains parenthesis, **DASM** will attempt to treat the specification as a ZIP archive member.

The –libexec option (provide a program to execute when the assembler cannot find a macro/copy member)

The *–libexec* option specifies a command to execute when the assembler has exhausted all of the specified library search paths and cannot find a copy or macro file. It is expected that the command will retrieve the specified file and place it somewhere in the library search path. Then, the assembler will try to find the file again, using the same library search path.

The *–libexec* option can be particularly helpful in situations where sources are not immediately available, but need to be retrieved from a source management system, or the mainframe host. For example, when an assembly begins, it may require the COPY file `MYCOPY` which is not present on a cross-platform host. The *–libexec* option may specify a program which fetches the `MYCOPY` file from the mainframe host, and places it in a known location. Then, **DASM** will retry the search, finding the `MYCOPY` member.

The –libncase option (use case-insensitive filename search)

The *–libncase* option directs **DASM** to use a case-insensitive search for filenames. It is only meaningful for files stored in Unix-style case-sensitive filesystems.

The way *–libncase* is implemented, the directory lookup proceeds in the usual (case-sensitive, depending on the OS) fashion. The first time a directory is seen, the list of files in it is read, then that list is used to enable case-insensitive searches. The list is cached, so the overhead is minimal for subsequent searches in that directory.

Listing and Print options

–pc=*control* –xpc=*control*	override/don't override certain print control statements in the source
–l / –xl	enable/disable generation of assembler listing
–esd / –xesd	enable/disable printing of symbol information in the listing
–fold / –xfold	enable/disable folding of lower-case letters to upper-case in the listing
–rld / –xrld	enable/disable printing of relocation information in the listing
–dx / –xdx	enable/disable the DSECT cross-reference information in the listing
–mx / –xmx	enable/disable the MACRO cross-reference information in the listing
–rx / –xrx	enable/disable REGISTER cross-reference information in the listing
–nx / –xnx	enable/disable UNREFS cross-reference information in the listing
–umap / –xumap	enable/disable USING MAP information in the listing
–uwarn *n* / –xuwarn	control emission of USING-related warnings
–cxs	enable the short cross-reference in the listing
–cxf	enable the full cross-reference in the listing
–xcx	disable the cross-reference in the listing
–os / –xos	enable/disable printing the option summary in the listing
–lc *count*	specify the number of lines per page in the listing
–lcond / –xlcond	enable/disable inclusion of conditional statements in the listing
–epops / –xepops	enable/disable expansion of macro operands in the listing
–term / –xterm	enable/disable error messages
–flisting=*filename*	specify an alternate name for the listing file

The –pc=*control*/–xpc=*control* option (override print control statements)

The *–pc=control* option is used to enable print control options. The *–xpc=control* option is used to enable/disable print control options.

The value of control is one of:

`ON`	Causes the assembler to produce a source listing.
`GEN`	Causes the assembler to include sources generated as the result of expansion of a macro.
`NODATA`	Causes the assembler to print only the first 8 bytes of constants in the object code. This option is deprecated in favor of the DATA option, but remains for backward compatibility.
`DATA`	Causes the assembler to print all of bytes of constants in the object code.
`MSOURCE`	Causes the assembler to include source statements generated during macro processing, as well as the assembled statements.
`UHEAD`	Causes the assembler to include the active `USING`s in the heading of each page in the listing.

Multiple *–pc control* and *–xpc control* options may be specified. Subsequent *–pc control* and *–xpc control* have a cumulative affect on the print control.

The print control values specified on the command line override any print control statements found in the source. Thus, if the source contained:

```
PRINT OFF
```

and the `-pc ON` option was specified, the `PRINT OFF` statement would be ignored, and the listing would contain subsequent instructions.

The –l/–xl option (enable/disable generation of the assembler listing)

The *–l*/*–xl* option controls generation of the listing. If *–xl* is specified, no listing file will be generated. The default is *–l*.

The –esd/–xesd option (enable/disable symbol information in the listing)

The *–esd/–xesd* options enable and disable inclusion of the Symbol Dictionary portion of the listing. The default value is *–esd*.

The –fold/–xfold option (enable/disable folding of lower-case letters to upper-case in the listing)

The *–fold/–xfold* option enables and disables the conversion of lower-case letters in the listing to upper-case.

If *–fold* is enabled, all the lowercase alphabetic letters will be converted to uppercase in every listing line.

–xfold is the default.

The –rld/–xrld option (enable/disable relocation information in the listing)

The *–rld/–xrld* options enable and disable inclusion of the Relocation Dictionary portion of the listing. The default is *–rld*.

The –dx/–xdx option (enable/disable the DSECT cross-reference in the listing)

The *–dx/–xdx* options enable and disable the generation of the `DSECT` cross-reference in the listing. The default is *–dx*.

The –mx/–xmx option (enable/disable the MACRO cross-reference in the listing)

The *–mx/–xmx* options enable and disable the generation of the MACRO cross-reference in the listing. The default is *–mx*.

The –rx/–xrx option (enable/disable REGISTER cross-reference information in the listing)

The *–rx/–xrx* options enable and disable the generation of REGISTER cross-reference information in the listing. The default is *–xrx*.

The –umap/–xumap option (enable/disable USING MAP information in the listing)

The *–umap/–xumap* options enable and disable the generation of USING MAP information in the listing. The default is *–umap*.

Previous versions of the assembler accepted *–ux/–xux* as synonyms of this option, but that usage is deprecated.

The –uwarn *n* and –xuwarn options (control emission of USING-related warnings)

The *–uwarn n* option controls the emission of diagnostics relating to common `USING` errors. *n* is a bitmask indicating which classes of warnings to report:

1 Any `USING` which cannot be used because it is completely replaced by another will trigger a `DASM300W`, `DASM301W`, or `DASM306W` warning.

2 A `DASM302W` is produced for any `USING` that is based on R0.

4 Ambiguous/overlapping `USINGs` generate `DASM303W` or `DASM306W`.

The default is *–uwarn 15*, which enables all of the warnings. *–xuwarn* has the same effect as *–uwarn 0*, disabling the warnings.

Note that if *–f NOUSING0* is specified then `DASM306W` will not be generated.

The –nx/–xnx option (enable/disable UNREFS cross-reference information in the listing)

The *–nx/–xnx* options enable and disable the generation of UNREFS cross-reference information in the listing, which lists all of the symbols defined in CSECTs which are not referenced. The default is *–xnx*.

The –cxs option (enable the short symbol cross-reference)

The *–cxs* option enables the generation of the short symbol cross reference. The short symbol cross reference only includes symbols that were actually referenced.

The –cxf option (enable the full symbol cross-reference)

The *–cxf* option enables the generation of the full symbol cross reference. The full symbol cross reference includes all symbols defined in the source.

The –xcx option (disables the symbol cross reference)

The *–xcx* option disables the generation of either the full or short cross reference. The default value is *–xcx*.

The –os/–xos option (enable/disable the options summary in the listing)

The *–os/–xos* options enable and disable the inclusion of the options summary at the beginning of the listing. The default is *–os*.

The –lc *count* option (define the number of lines per page)

The *–lc count* option defines the number of lines printed on a page before moving to the next page. The default value is 55 lines per page.

The –lcond/–xlcond option (include/omit conditional statements in the listing)

The *–lcond/–xlcond* options cause the assembler to include or omit conditional statements in the listing. When *–xlcond* is specified, the assembler will not include statements which are suppressed by a conditional evaluation. The default is *–xlcond*.

The –epops/–xepops option (include/omit expanded macro operands in the listing)

The *–epops/–xepops* option causes the assembler to include or omit expanded macro operands in the listing. The default is *–xepops*.

The –term/–xterm option (enable/disable error messages)

The *–term* option causes the assembler to write error messages to the STDERR file stream on cross-platform hosts or the SYSTERM DD on z/OS and CMS hosts.

Note that the **DASM**-produced banner and summary messages are written to the STDERR stream and STDERR DD as well. The *–quiet* option can suppress those messages.

–term is equivalent to the HLASM `TERM` option.

The –flisting=*file* option (specify the name of the listing file)

The *–flisting=file* option instructs the assembler to use the given file name for the generated listing file. When *–flisting=file* isn't specified, the assembler uses the default name for producing the listing. On cross-platform hosts, the default listing file name is based on the input file name, with a suffix of `.lst`. On z/OS, the default listing file name is the `SYSPRINT` DD.

Miscellaneous options

–fmesg=*style*	specify the style of error messages
–M[=*filename*]	generate a file dependency list
–A[=*filename*] / –xA	enable/disable generation of a separate ADATA information file
–fadver=*version*	specify which format version for ADATA information
–fadftp	output block mode FTP markers in the ADATA file
–fadrdw	output RDW headers in the ADATA file
–E=*filename*	specify an alternative file to log error messages
–fevents=*filename*	emit an IBM-compatible events listing
–options=*options string*	specify options in HLASM-style syntax

The –fmesg=*style* option (specify the style of messages)

The *–fmesg=style* option instructs the assembler to use a different style of formatting error messages written to `stderr`. This does not alter the style of messages that appear in the listing.

The supported styles are:

`dasm` Default style on z/OS and UNIX hosts. Messages are formatted with the text "`dasm:`" appearing initially on the line. This style can make looking for errors in large build logs easier.

`dasmx` Extended `dasm` format. Extended `dasm` format will cause an extra message (number DASM435I) to be generated when a message is generated in a macro. This extra message indicates the source line where the macro was invoked.

`gcc` Use the gcc-style of messages, which can be helpful when using **DASM** with tools on UNIX platforms, such as `emacs`.

`gccx` Extended `gcc` format. Extended `gcc` format will cause an extra message (number DASM435I) to be generated when a message is generated in a macro. This extra message indicates the source line where the macro was invoked.

`microsoft` Default style on Windows hosts. Messages are formatted so they can be recognized by Microsoft's Visual Studio environment.

The –M[=*filename*] option (generate file dependency list)

The *–M[=filename]* option causes the assembler to output dependency rules suitable for the `make` program. These rules describe the dependencies of the assembled file, indicating that to recreate the target object file, the given sources and any macros used by this assembly are required. By default, the dependency information is written to the `stdout` stream. The optional *=filename* can be used to specify a particular file to write the dependency information.

The depencies will be listed in the form

target: *source*

where *target* is the name of the object file being generated, and *source* is the source file being assembled. If there are any dependent macros being used, other lines will be generated that list the dependent macros as *source*s.

If no object file is being created, a name ending in `.o` will be automatically generated for the dependency.

The –A[=*filename*] and –xA options (enable/disable generation of a separate ADATA information file)

When *–A=filename* is specified, the assembler will write any ADATA information to the specified file. The assembler generates ADATA information compatible with IBM's HLASM assembler. This information includes data describing the source that was assembled and is used by several runtime debuggers. The assembler also places any information from ADATA pseudo-operations in the specified file.

If the optional *=filename* isn't specified, the assembler places the information in the file that's named the same as the source file, with any extension replaced with the extension "`.dat`".

Note that when generating GOFF-style object files, it is possible to embed this ADATA information directly in the generated object via the *–goffadata* option.

The –fadver=*version* option (specify which format version for ADATA information)

The format of ADATA information differs between HLASM V1R4 and HLASM V1R5. The *–fadver=version* option selects which version **DASM** will generate.

version can either be 4 or 5.

By default, **DASM** generates ADATA information compatible with HLASM V1R5.

Note that on z/OS, the ADATA resides in a variably blocked (`RECFM=VB`) file with a block size of 32760 (`BLKSIZE=32760`) and record length of 32756 (`LRECL=32756`). Also, on z/OS, the *filename* specified is in Dignus file name format. For example, `-A=//DDN:ADATA` would specify that the ADATA information is to be written to the DD named "`ADATA`".

The –fadftp option (output block mode FTP markers in the ADATA file)

The *–fadftp* option tells **DASM** to add block mode FTP markers to the generated ADATA file. These markers are interpretted by the FTP server to enable the mainframe to construct proper-length VB records. Use this option if you are running **DASM** on a non-mainframe host (such as a PC) but you are going to use the ADATA as a VB dataset on the mainframe. Note that the ADATA files generated when *–fadftp* is enabled are not suitable for use on the PC or for use in FB datasets.

When you transfer the generated ADATA file to the mainframe, you must issue a `quote mode b` command in your FTP client. This will instruct the FTP server on the mainframe to expect block mode data. Otherwise the markers will appear as if they were a part of the ADATA, essentially corrupting the file.

The –fadrdw option (output RDW headers in the ADATA file)

The *–fadrdw* option tells **DASM** to add an RDW header to each record of the ADATA file. The RDW header is the format used to identify blocks in `VB` encoded files. The header consists of a two-byte value indicating the length of the record (including the header bytes) followed by two zero bytes. Note that the ADATA files generated when *–fadrdw* is enabled are not suitable for use on the PC or for use in FB datasets.

The –E=*filename* option (specify an alternative file to log error messages)

The *–E=filename* option causes the assembler to write any error messages to the specified *filename*. Every message that is typically logged to the `stderr` file descriptor will instead be directed to the named file.

The –fevents=*filename* option (Emit an IBM-compatible events listing)

The *–fevents=filename* option causes **DASM** to generate an event listing in the named file. Several IBM products use event listings of this format to communicate error message information between compilers and user interfaces. Using this option, you may generate an events file for use with any products that share this format.

The events file contains 3 types of single-line records:

`ERROR 0` *A* `0 0` *B* `0 0 0 DAS`*D E F G H*

A The number of the file where the error occurred.

B The record number at which the error occurred.

D The error code.

E A severity, one of `I` for informational messages, `W` for warnings, `E` for errors, `S` for severe errors, or `U` for unrecoverable errors.

F The mainframe return code for the error.

G The length of the error message.

H The error message.

`FILEID 0` *A* `0` *C D*

A The number of the file.

C The length of the file name.

D The file name.

The –options=*options string* option (Specify options in HLASM-style syntax)

For easier migration from HLASM, the *–options=* option can specify a string that is processed as an HLASM options parameter.

The values specified as *options string* are the options available for passing to HLASM via the `PARM` specification in JCL.

Current supported HLASM options include:

ADATA, XOJECT, GOFF, LIST, OBJECT, LINECOUNT, LANGUAGE, SIZE, SYSPARM, TRANSLATE, ALIGN, ASA, BATCH, CODEPAGE, COMPAT, ESD, FLAG, FOLD, LIBMAC, MXREF, OPTABLE, NOPCONTROL, PCONTROL, PROFILE, RA2, RENT, RLD, RXREF, SECTALGN, THREAD, USING, XREF, SUPRWRN

The following limitions and differences apply between **DASM** and HLASM:

- LANGUAGE is limited to LANGAUGE(UE) or LANGUAGE(ES).
- **DASM** doesn't limit the amount of memory allocated at assembly time, so the only meaningful values of the SIZE option are SIZE(0) and SIZE(MAX). Any other integer is silently accept and SIZE(MAX) will be applied.
- TRANSLATE is limited to TRANSLATE(AS) or NOTRANSLATE.
- ASA, CODEPAGE, DBCS, DECK, DXREF, FOLD, LIBMAC, NOPCONTROL, and TEST are accepted but do nothing.
- ESYM examines environment variables.

  Unlike HLASM, the ESYM builtin SETC function queries the runtime environment. On non-z/OS hosts, that is the only environment available. On z/OS, if no value is found in runtime environment then the lookup proceeds by looking for the JCL system value for the given name.

  Thus, when running under batch (where no environment is provided) DASM will act as HLASM does, and query JCL symbols.

  However, when running under USS, environment variables inherited from the caller will be available and examined before searching the JCL symbol space.

  HLASM does not support symbol look up in the environment, it only examples JCL system symbols.

DASM also has extended options used to offer controls that are only available in **DASM**. These include:

MAPAT/NOMAPAT Enable or disable the *–fmapat* option.

DUPALIAS/NODUPALIAS Enable or disable the *–fdupalias* options.

IDR(*value*) Set the END card IDR value. For more information see the *–fidr* option.

DWARF(*filename*) Set the output DWARF debugging information file. For more information, see the *–fdwarf* option.

Linking Assembled objects

Once the assembly source has been assembled, it can be linked using the typical link procedures. If **DASM** was executed on a cross-platform host, the resulting object decks should be transferred to mainframe host environment via FTP or some other binary-mode transfer mechanism.

ADATA Information

DASM will optionally generate Associated Data, or ADATA information, either as a separate file, or embedded within a GOFF format object file. The *–A* and *–goffadata* options control the generation of ADATA information.

ADATA information contains descriptions of the source that was assembled, and can be used by debuggers or cross-referencing tools. Also, the ADATA pseudo-op can be used to augment the generated ADATA information. The IBM system macro ASMADATA provides a definition of the various ADATA record formats.

For more information about ADATA and the ADATA record formats, see the IBM publication *High Level Assembler for MVS & VM & VSE Programmer's Guide, Release 5.*

Differences between DASM and HLASM ADATA information

DASM provides ADATA information compatible with HLASM V1R5, with the following differences:

Source Analysis Records

- `Directive` and `Macro` calls which support alternative continuations do not have substitution performed on the output record data. As implemented in HLASM V1R4, the substitution was unreliable, for instance, comments can overlay operands. Rather than emulating the unreliable output, **DASM** provides the raw source lines.
- **DASM** correctly reports continuation lines as the same type as the first line of the statement.
- **DASM** does not substitute 'X' for the original continuation character in many of the situations that HLASM does.

- **DASM** correctly reports `Macro Call` continuation lines as continuation lines in instances where HLASM does not.
- `File Number` and `Parent File Number` are correctly reported when macros are nested.
- **DASM** always reports the record number for continuation lines as the first record in the statement. HLASM sometimes reports the first record and other times reports the last record.
- For `Macro Calls`, HLASM only sets the `Location Counter` if the macro call is not made from the original source file. **DASM** always sets the `Location Counter`.

Symbol Records

- The type "`*-in-Literal Name`" is reported as a `Literal Name`.
- **DASM** does not assign an ESD ID to an absolute `EQU` type symbol.

Symbol XREF Records

- For all symbols, **DASM** writes `XREF` records directly after the corresponding `Symbol Records`. This varies from HLASM which writes `XREF` records for referenced symbols following the corresponding `Symbol Record`, and records for un-referenced symbols after all `Symbol Records` have been written.
- **DASM** consistently reports all symbols which are used to generate a modified location as `Modified`, while HLASM does not.

Unexpected or undocumented HLASM behavior supported by DASM

There are several situations where HLASM either does not conform to the HLASM documentation, or behaves in an unexpected fashion. **DASM** mimics these situations to allow for full interoperability between HLASM and **DASM** generated information.

The Dignus CICS Command Processor, DCCPA

Distributed with the assembler is the Dignus CICS Command Processor, **DCCPA**. For assembler source which uses `EXEC CICS` commands, **DCCPA** is used to translate the `EXEC CICS` commands into normal assembly code prior to invoking **DASM**. This is especially useful in cross environments where IBM's translators cannot be used. For further information about CICS, see the IBM publication *CICS Transaction Server for z/OS: CICS Application Programming Reference* document number SC34-5994-02.

Running DCCPA

On Windows and Unix, **DCCPA** is executed with a command line of the form

`dccpa` *[options] [input file]*

If no input file is specified, input is read from **stdin**.

When run on z/OS, options may be specified in the `PARM` statement. If no input file is specified, the `SYSIN` DD is used. Output defaults to `SYSPUNCH`, which would typically be used as input for **DASM** in the next step. Informational and error messages are output to `STDOUT` and `STDERR` DDs. The following JCL could be used on z/OS:

```
//DCCPA JOB
//DCCPA  EXEC PGM=DCCPA,PARM='options'
//STEPLIB DD DSN=DIGNUS.LOAD,DISP=SHR
//STDERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSIN    DD *
     <assembler source>
```

DCCPA Options

Options for **DCCPA** are summarized in the table below.

–A	process assembly source
–C	process C source
–o *file*	place translated output in the file named *file*
–fdli / –fnodli	enable/disable support for `EXEC DLI`
–fgds / –fnogds	enable/disable support for GDS commands
–fsp / –fnosp	enable/disable support for System Programmer commands
–fcols=*n*	the output in C mode will be *n* columns
–fseq	the output in C mode will include sequence numbers
–fmrc / –fnomrc	enable/disable use of mainframe-style return code
–fflag=*code*	output only error messages of a certain priority
–fepilog / –fnoepilog	enable/disable use of `DFHEIRET` macro
–fprolog / –fnoprolog	enable/disable use of `DFHEISTG`, `DFHEIEND`, and `DFHEIENT` macros
–ferrlist / –fnoerrlist	enable/disable listing errors on *stderr*
–fexci / –fnoexci	enable/disable EXCI mode
–fleasm / –fnoleasm	enable/disable LE ASM compatibility
–fvse / –fnovse	enable/disable VSE compatibility
–fedf / –fnoedf	enable/disable EDF mode

The –A option (process assembly source)

The *–A* option tells **DCCPA** to process assembly source code. *–A* is the default for **DCCPA**.

The –C option (process C source)

The *–C* option tells **DCCPA** to process C source code instead of assembly code.

The –o *file* option (specify the name of the output file)

The *–o file* option specifies that the translated output should go to a file other than the default. On Windows and Unix systems, the default is `ccp.out`, while in z/OS it is the `SYSPUNCH` DD.

The –fdli and –fnodli options (enable/disable `EXEC DLI`)

The *–fdli* option enables support for `EXEC DLI` statements as well as `EXEC CICS` statements.

The –fgds and –fnogds options (enable/disable GDS commands)

The *–fgds* option enables support for GDS commands (commands of the form `EXEC CICS GDS` ...). The default is *–fnogds*.

The –fsp and –fnosp options (enable/disable System Programmer commands)

The *–fsp* option enables support for System Programmer commands. The default is *–fnosp*.

The –fcols=*n* option (specify column width)

When processing C source code the *–fcols=n* option instructs **DCCPA** to use only *n* columns in its output. Assembly code is always limited to 72 columns with continuations.

The –fseq option (generate sequence numbers)

When processing C source code the *–fseq* option causes **DCCPA** to generate sequence numbers in the output. This implicitly sets `-fcols=72`. The sequence numbers appear in columns 73-80.

The –fmrc and –fnomrc options (enable/disable mainframe-style return codes)

The *–fmrc* option specifies that the translator should use mainframe-style return codes to indicate the exact error level reached. The *–fnomrc* option specifies that

the translator should use Unix-style return codes that are either 0 (success) or -1 (errors). On z/OS *–fmrc* is the default, while on cross-platform hosts *–fnomrc* is the default.

The –fflag=*code* option (output only error messages of a certain priority)

The *–fflag=code* option specifies that only error messages at least as severe than *code* should be displayed. Valid values for *code* are `I` for informational messages, `W` for warnings, `E` for errors, and `S` for severe errors. The default is *–fflag=I*.

The –fepilog and –fnoepilog options (enable/disable use of `DFHEIRET` macro)

The *–fnoepilog* option specifies that the `DFHEIRET` macro should not be invoked in the translated assembly source, This option is needed to make the CICS `RETURN` command effective. The default is *–fepilog*.

The –fprolog and –fnoprolog options (enable/disable use of `DFHEISTG`, `DFHEIEND` and `DFHEIENT` macros)

The *–fnoprolog* option specifies that the `DFHEISTG`, `DFHEIEND` and `DFHEIENT` macros should not be invoked in the translated assembly source. These macros define local storage that is allocated a program start up. The default is *–fprolog*.

The –ferrlist and –fnoerrlist options (enable/disable listing of errors on *stderr*)

The *–ferrlist* option enables the listing of errors on *stderr*. The *–fnoerrlist* option specifies that errors should just be listed as comments in the translated output file and is the default.

The –fexci and –fnoexci options (enable/disable EXCI mode)

The *–fexci* option instructs **DCCPA** to run in EXternal Call Interface mode, for processing files which contain only a special form of the `EXEC CICS LINK` command. When run in EXCI mode, no other commands will be translated. Outside of EXCI mode, this special form of the `EXEC CICS LINK` command is unavailable.

The –fleasm and –fnoleasm options (enable/disable LE ASM compatibility)

The *–fleasm* option changes some macro parameters to create a Language Environment conforming assembler program, rather than one to be loaded in the CICS environment. It should only be used in assembler mode (*–A*). The default behavior is *–fnoleasm*, to generate a program to be executed from the CICS environment.

The –fvse and –fnovse options (enable/disable VSE compatibility

By default, **DCCPA** generates output compatible with the z/OS or MVS versions of the IBM CICS preprocessor. However, the CICS preprocessor for VSE supports a few new commands (`SPOOLCLOSE REPORT`, `SPOOLOPEN ESCAPE`, `SPOOLOPEN MAPNAME`, `SPOOLOPEN REPORT`, `SPOOLOPEN RESUME`, `SPOOLWRITE MAPNAME`, and `SPOOLWRITE REPO`), and has alternative translations for a few others (`INQUIRE PROGRAM` and `INQUIRE TASK`). When *–fvse* is supplied on the commandline, **DCCPA** will generate translations compatible with the VSE preprocessor.

The –fedf and –fnoedf options (enable/disable EDF mode)

The default is *–fedf*. If *–fnoedf* is specified then the `X'40'` bit in the common flags area of each `DFHECALL` invocation is set.

Writing Linux/390 and z/Linux programs

DASM can take existing assembly language text and assemble it for running on the Linux/390 or z/Linux operating systems.

The goal for this feature is to be able to move existing programs to Linux/390 or z/Linux as transparently as possible. With some caveats this can be easily accomplished in many situations.

Linux features

ELF object format

The object file format on Linux/390 and z/Linux is not the traditional OBJ-style file format used in many mainframe operating systems.

Linux uses the *Executable and Linking Format* or `ELF` file format. For information regarding the `ELF` file format, consult the `System V Application Binary Interface`. There are also several web resources which provide information on the `ELF` format.

DASM with the *–flinux* or *–flinux64* option will produce `ELF` format objects suitable for use in the Linux/390 and z/Linux environments.

ASCII character constants

Character constants in Linux are ASCII, not EBCDIC. When the *–flinux* option is enabled, **DASM** will generate ASCII character constants. Note that ASCII character constants can be independently enabled with the *–fasciiout* option as well.

Section management

Sections in ELF format objects are divided into classes, which are dissimilar from the traditional CSECT approach. However, **DASM** will transparently handle that in many situations, avoiding changes to existing source.

Typically, an ELF program consists of a .text class and a .data class. .text contains the executable instructions that comprise the program, while .data contains initialized program data.

By default, CSECTs are placed in the .text section consecutively as they appeared in the source file. This is the executable code section for the resulting program.

CSECTs are aligned on the CSECT boundary within each assembled source file. However, each .text section for each object is combined into a single .text section in the resulting program by the linker, ld. Thus, alignment between object files is the responsibility of the ld linker. Consult the ld documentation for options controlling object alignment.

If the CSECT does not contain executable code, but instead is meant to be program data, it can be directed to the .data section. To cause a CSECT and its assembled bytes to appear in the .data section, use the **DASM**-specific B_DATA CATTR instruction within the section. This indicates that the CSECT should be placed into the .data section when the resulting ELF object is generated.

The B_DATA CATTR instruction indicates that the enclosing CSECT belongs in the .data section.

For example:

```
DATA CSECT
B_DATA CATTR
VAR  DC F'1'
    ...
```

defines a CSECT named 'DATA', and places the value F'1' in that CSECT at the label VAR. These bytes will not appear in the .text ELF section, but rather in the .data section because of the B_DATA CATTR statement.

Multiple B_DATA CATTR specified CSECTs will be concatenated into the single .data section in the resulting ELF object.

Debugging under Linux

If the *–g* option is specified on the **DASM** command line, **DASM** will generate debugging information typical of Linux implementations.

This information includes line number information which reflects the line number of the original source or macro file that generated the code, as well as information that describes any DSECTs defined in the source.

DSECTs will appear to the Linux debugger as C structures, with appropriate types.

Typically, the GNU **gdb** debugger is installed on Linux platforms. **gdb** debugging of **DASM**-generated objects works quite well. Using **gdb** the programmer can single-step through the source, set break points, view/alter data, view DSECTs, etc. For more information on how to use `gdb`, consult the Linux documentation.

Differences with traditional programs

AMODE 24 and RMODE

Linux/390 and z/Linux do not support AMODE 24 and/or RMODE 24 programs. Thus, any AMODE-dependent code needs to be altered to consider this. For this reason, any address constants that are not 4 or 8 bytes will be flagged as an error.

Similarly, `DASM` will issue a warning should it encounter any AMODE/RMODE statements.

Q-type constants and DXDs

When *–flinux* is specified, Q-type constants are treated as references to the ELF `.bss` section. Also, `DXD` statements cause definitions in the `.bss` section. The `CXD` relocation is not supported.

The `.bss` section contains uninitialized data, which is allocated and filled with zeros when the Linux program begins execution. Thus, it is very similar to the Pseudo-Register Vector (`PRV`) which Q-type constant reference and DXDs define.

Typically, a traditional program uses the `CXD` relocation to determine the size of the linker-defined `PRV`. It then allocates enough source for that size and retains a pointer to the allocated storage in a known location.

Then, when accessing elements of the `PRV`, the traditional program would retrieve the pointer to the `PRV` and add the offset specified in the Q-type constant, resulting in the address of the desired datum.

When programming for Linux/390 - all that needs to change is the allocation portion, as the `.bss` section is automatically allocated. So, when moving this type code to a Linux environment, simply remove the allocation routine, and set the value of the "`PRV`" to zero. When the zero-value is later added to the Q-type constant value, it results in the proper address in the `.bss` section for the datum.

Function linkage and parameters

Linux function linkage is quite different from the traditional linkages employed by most assembly programs. Your Linux distribution should contain more details regarding function linkage and parameter passing.

Lower-case identifiers

Typically, Linux functions and data, particularly those made available in the distributed C function libraries, use lower-case names.

The traditional assembler approach is to gratuitously upper-case names written in the output object deck. `DASM` continues to follow that approach for compatibility with older, existing code.

To properly access functions and data from the Linux C libraries, the `ALIAS` instruction should be used. In the following example, we are accessing the `printf()` function externally from this source:

```
printf ALIAS C'printf'
       ...
     LA 2,message
     L 3,=V(printf)
     BASR 14,3   Call printf(message)
```

Entry point

Linux programs always begin at the function named "`main`". Linux programs do not support multiple entry points.

Using **DASM** it is straight-forward to define a CSECT named "`main`" and invoke a desired alternate entry point, e.g.:

```
main  CSECT
main  ALIAS C'main'
*
* function prologue
*
     STM 6,15,24(15)
     BASR 13,0
     USING *,13
     LR   1,15
     S    15,FRAMESIZE
```

```
     ST   1,0(15)
     LR   11,15
*
* Branch to the entry point named "MYEP"
*
     L 3,=V(MYEP)
     BASR 14,3

*
* main() epilogue.. R2 contains return code.
*
     L   4,176(15)
     LM  6,15,144(15)
     BR  4
     END
```

System facilities

Linux does not provide the operating system defined macros typically used on z/OS or other traditional mainframe environments. Instead, the operating system interface functions are defined in the C programming library.

Thus, I/O, memory allocation, and other system functions should be accessed through the typical C library functions.

Portions of existing programs which use z/OS or other facilities will need to be rewritten to operate on Linux.

Example 31-bit Linux/390 programs

The following program is a simple "Hello-world" program in 31-bit AMODE. It writes "`Hello world.`" to the STDOUT stream by invoking the C library's `printf()` function. (Note that for 64-bit programs, the prologue, epilogue, call sequence, instructions, etc... would be different.)

```
main  CSECT
main  ALIAS C'main'
printf ALIAS C'printf'
*
*  Typical Linux prolog
*  R15 is the save area (stack pointer).
*  The definition of FRAMESIZE should be enough
*  to contain any local variables needed in this
*  stack instance.
```

```
*
     STM 6,15,24(15)
     BASR 13,0
     USING *,13
     LR   1,15
     S    15,FRAMESIZE
     ST   1,0(15)
     LR   11,15
*
*  Call printf()
*
     LA 2,HELLO
     L 3,=V(printf)
     BASR 14,3
*
*  Set our return code to 0
*
     LA 2,0  Return code
*
*  Return to the caller
*
     L   4,176(15)
     LM  6,15,144(15)
     BR  4
FRAMESIZE  DC F'120'
*
* Define a C-style null-terminated
* string "Hello world.\n".  The
* \n character is X'0A'.
*
HELLO DC C'Hello world.',X'0A',X'0'
     END
```

In the following example, the program reads the incoming `argc` and `argv` arrays and writes the values out, along with a counter to the `stdout` stream.

The counter is a datum contained in the `.data` section because it is declared in a CSECT that has the `B_DATA CATTR` instruction.

```
main  CSECT
main  ALIAS C'main'
printf ALIAS C'printf'
     STM 6,15,24(15)
     BASR 13,0
     USING *,13
     LR   1,15
     S    15,FRAMESIZE
```

```
     ST   1,0(15)
     LR   11,15

*
*  argc is in R2 (the first incoming parameter)
*  argc is in R3 (the second incoming parameter)
*  save these in 8 and 9 respectively
*
    LR 8,2
    LR 9,3

*
*  initialize ‘counter’ to 1
*
   L  4,=V(counter)
   LA 5,1
   ST 5,0(0,4)
*
*  Loop while ‘counter’ is < argc (R3)
*
startloop DS 0H
   L  5,0(0,4)
   CR 5,8
   BNL endloop

*
* invoke printf("%d: %s\n",counter, argv[counter])
*
   LA 2,format
   LR 3,5    current value of counter
   LR 6,5
   SLL 6,2   counter * 4
   L   4,0(6,9)   argv[counter]
   L   7,=V(printf)
   BASR 14,7
*
*  Increament counter and branch back
*
   L   4,=V(counter)
   L   5,0(0,4)
   AHI 5,1
   ST  5,0(0,4)
   B  startloop
endloop DS 0H


*
```

```
*  Set our return code to 0
*
     LA 2,0  Return code
*
*  Return to the caller
*
     L   4,176(15)
     LM  6,15,144(15)
     BR  4
FRAMESIZE  DC F'120'
format DC C'%d: %s',X'0A',X'00'

*
* Define the CSECT for .data
*  (note that it can be an unnamed CSECT)

     CSECT
B_DATA CATTR        indicate this is a .data section
     ENTRY counter  make 'counter' externally visible
counter DC F'0'
     END
```

HLASM asma90 compatibility

DASM can operate in a fashion that is compatible with the IBM "High Level Assembler for Linux on zSeries".

IBM provides the HLASM assembler in this environment, with the program named `asma90`. If the DASM executable is invoked via that name, it operates in "asma90 mode".

To invoke **DASM** with the `asma90` name, simply copy the **DASM** executable to one named `asma90`. In determining if "asma90 mode" should apply, **DASM** ignores any suffix, so a name of `asma90.linuxz` would trigger "asma90 mode" as well as just `asma90`.

In "asma90 mode", **DASM** accepts most parameters that the HLASM `asma90` program accepts, and processes them in a similar fashion.

"asma90 mode" is not limited to the z/Linux host platform. The assembler can operate in this fashion on any host plaform except native z/OS or CMS.

Invocation parameters

The "asma90 mode" invocation options are:

-a	adata file name
-l	listing file name
-o	output object file name
-t	error output (term) file name
-L	library search specification template
–options='*options string*'	HLASM options

The `asma90` program also provides a `-E` option which is not supported by **DASM**.

If `-l` is not specified, the `LIST` option is disabled. If `-o` is not specified, the OBJECT option is disabled.

If `-t` is not specified, error messages are written to the STDOUT descriptor.

Since HLASM cannot generate `XSD`-style object files the default is for `ESD`-style object files, unless the `GOFF` option is specified in the —*options* string.

Library search rules

The `asma90 -L` option names a pathname library search template. In this template, the '`*`' character is replaced with the name of the macro or copy member being included. Multiple templates can be specified in one `-L` option, separated with a colon.

When searching for macro or copy files, the HLASM `asma90` assembler expands the template, replacing '`*`' with the name of the macro or copy member; first trying the name in upper case and then in lower case.

In **DASM**, the `asma90 -L` specifications are translated into `DASM` specifications by generating two `DASM` library search templates for each `asma90` template. In the first of these, any '`*`' is replaced with `&M` for the upper-case search, in the second it is replaced with `&m` for the lower-case search.

For example, the `asma90` option `-L mydir1/*.mac:mydir2/*.EXT` is translated into the **DASM** templates `mydir1/&M.mac`, `mydir1/&m.mac`, `mydir2/&M.EXT` and `mydir2/&m.EXT`.

This translation effectively mimics the HLASM `asma90` behavior.

The **DASM** listing will reflect this translation, and not the original `-L` options.

Multiple `-L` options are also allowed.

Listing file

In the `asma90` program, the listing file is generated as if HLASM was executing under z/OS. The `asma90` listing file is a fixed width EBCDIC file with no new-line delimiters.

Therefor, users of `asma90` are required to translate this file to view it in an ASCII setting.

DASM does not mimic this behavior when executed as `asma90`. Instead the listing file will be in the native format (ASCII in most environments) with the proper new-line delimiters.

No user translation of the listing file is required.

ASCII/EBCDIC translation

By default **DASM** uses a variant of IBM1047 code page for its translation between ASCII and EBCDIC. This has several advantages, being the same mapping that IBM uses for it's `iconv` tables, and other utilities on Unix Systems Services.

The HLASM `asma90` program uses the IBM037 code page.

The primary difference is these two code pages is the mapping of the square bracket characters.

When operating in "asma90 mode", **DASM**'s internal tables are altered to match IBM037 so that any translation of `PUNCH` or `REPRO` data will match the output from `asma90`.

z/TPF use

z/TPF's build process uses a facility named '`maketpf`'. To use the Dignus assembler in this context, it should be named 'asma90' to match what '`maketpf`' expects.

Because of the 'asma90' compatibilty, very little in '`maketpf`' needs to be changed.

However, the '`maketpf`' build rules invoke the IBM-supplied '`calst`' program. '`calst`' converts the IBM EBCDIC assembler listing to ASCII for presentation to the user. Because the Dignus assembler in 'asma90' mode already produces an ASCII listing, this step is not required.

The `maketpf.rules_set_programs` file contains the definition of the `CALST` variable. Simply change that from the existing `calst -f IBM037 -t IBM819` to the value `touch`, to invoke the `touch` command in place of `calst`.

Assembler messages

DASM produces messages in the listing file similar to IBM's HLASM product. **DASM** also writes messages to the STDERR file stream on the console.

Although the text of **DASM** messages may be slightly different, almost all message numbers and descriptions correspond to the HLASM message numbers, so the IBM document "High Level Assembler for z/OS & z/VM & z/VSE Programmer's Guide" may also be a useful reference for information about a particular message.

The following diagnostic messages are unique to DASM and do not correspond to an equivalent HLASM message: DASM900W DASM901E DASM902E DASM903W DASM909W DASM911E DASM913W DASM914S.

Message Format

DASM produces diagnostic messages in the following format:

`DASM`*nnns*

where *nnn* is a three-digit message number and *s* is the severity indicator. Message numbers correspond with HLASM message numbers whenever possible.

The severity indicators correspond to the following codes:

`I` - Informational
: This character indicates a severity code of 0. The message is informational only and the assembly will continue.

`N` - Notice
: This character indicates a severity code of 2. The assembler is noting certain conditions which may be meaningful to your program.

`W` - Warning
: This character indicates a severity code of 4, a warning. The assembler has discovered a situation which could potentially cause problems when your program is executing.

`E` - Error
: This character indicates a severity code of 8, an error. The assembler has detected an error situation, but will continue to try and process the source as best as is reasonably possible.

`S` - Severe
: This character indicates a severity code of 12, a severe error. In this situation, the assembler will produce a zero for any erroneous instructions.

`C` - Critical
: This character indicates a severity code of 16, a critical error. The program will not assemble.

`U` - Unrecoverable
: This character indicates a severity code of 20 or greater, an unrecoverable error. The assembler has detected a situation from which it cannot recover and assembly will be halted.

Messages

The following list describes the messages produced by **DASM**.

DASM001E Operation code not allowed to be generated
: The source attempted to produce a restricted opcode from a macro variable substitution.

DASM002S Generated statement too long; statement truncated - xxxxx
: The statement produced by a macro definition expansion exceeded the assembler limits.

DASM003E Undeclared variable symbol; default=0, null, or type=U
: A variable symbol was used as an operand without being declared. The symbol is given an appropriate default value.

DASM004E Duplicate SET symbol declaration; first is retained - xxxxx
: A SET symbol was declared more than once. The first declaration is used. Note that a SET symbol is declared when it is the name of a SET statement, an operand of an `LCL` or `GBL` statement, or in a macro prototype statement.

DASM005S No storage for macro call; continue with open code
: The assembler exhausted the available storage when processing an inner macro call. The assembler will attempt to continue the assembly process with the next open code statement.

DASM007S Previously defined sequence symbol - xxxxx
: The given sequence symbol has been previously defined in the name field of a prior statement.

DASM008S Previous defined symbolic parameter - xxxxx

The given symbol was previously used as a symbolic parameter.

DASM009S System variable symbol illegally re-defined

The name field of macro prototype statement uses a system variable name.

DASM010E Invalid use of symbol qualifier - xxxxx

The given qualifier was incorrectly applied, or applied to an undefined symbol.

DASM011E Inconsistent global declarations; first is retained - xxxxx

The given global SET symbol has been defined more than once, and the multiple definitions are inconsistent and type or dimension.

DASM012E Undefined sequence symbol; macro aborted - xxxxx

The given sequence symbol is used as an operand but has not been defined.

DASM013S ACTR counter exceeded - xxxxx

The loop counter used by the assembler has been set to 0. This counter is decremented each time an AIF or AGO branch is correctly processed. This message potentially indicates an AIF/AGO loop or other macro issue.

DASM014E Irreducible qualified expression

The statement cannot be assembled because at least two qualified symbols are used in a complex relocatable expression, or at least two symbols with different qualifiers are paired in an absolute expression.

DASM015W Literal bounds exceeded

The address expression resolves to an address outside of the bounds of a literal, or the length of the receiving field of the literal is longer than the literal.

DASM016W Literal used as the target of instruction

The target of the instruction is a literal which indicates a potential error.

DASM017W Undefined keyword parameter; default to positional, including keyword - xxxxx

The given keyword parameter is not in the corresponding macro prototype statement. This can also occur if a positional parameter contains an equals sign (=).

DASM018S Duplicate keyword in macro call; last value is used - xxxxx

The given keyword appears more than once.

DASM020E Illegal GBL or LCL statement - xxxxx

No operand was specified for the `GBL` or `LCL` statement.

DASM021E Illegal SETB/AIF statement - xxxxx

The operand of a SETB statement is not 0, 1, or the operand of an AIF is not a boolean expression, or there is an issue with the parenthesis in the SETB or AIF statement.

DASM023E Symbolic parameter too long - xxxxx

The given symbolic parameter is longer than 63 characters, including the ampersand.

DASM024E Invalid variable symbol - xxxxx

The given symbol is not a valid symbolic parameter or SET symbol.

DASM025S Invalid macro prototype operand - xxxxx

The given symbol is not a valid operand in a macro prototype statement.

DASM026S Macro call operand too long; operand truncated

The macro operand is too long, macro operands are limited to 255 characters.

DASM027S Excessive number of operands

Too many operands were specified on the statement.

DASM028E Invalid displacement

The displacement specified in an address, or in an S-type address constant is not in the range 0 to 4095 inclusive.

DASM029E Incorrect register or mask specification - xxxxx

The given value is invalid either as a register or as the mask in an instruction.

DASM030E Invalid literal usage - xxxxx

The specified literal is incorrectly used in an instruction or another literal.

DASM031E Invalid immediate field

The specified field is not within the proper limits for the immediate field of the instruction.

DASM032E Relocatable value found where absolute value requested

A relocatable expression was used where an absolute value is required, or an expression based on a DSECT is used when the expression requires a storage address.

DASM033I Storage alignment unfavorable

One or more addresses referenced by the instruction may not be optimally aligned for the best performance on certain hardware.

DASM034E Operand operand *beyond active USING range by* xxxxx *bytes*

A active USING was discovered for *operand* but the operand's address falls outside of the active USING's address range.

DASM035S Invalid delimiter - xxxxx

A required delimiter is either missing or incorrect.

DASM036W Reentrant check failed

The assembler has detected an instruction which may store directly into a CSECT or common area.

DASM037E Illegal self-defining value - xxxxxx

A self-defining term in an expression, either decimal, binary, hexadecimal or character, contains an illegal character or invalid format.

DASM038S Operand value falls outside of current section/LOCTR

An `ORG` statement specifies a location that is not within the current section or LOCTR in which the `ORG` statement is used.

DASM039S Location counter error

The assembly sources specifies a section that is larger than the allowable size for ESD/XSD style objects, X'FFFFFF' bytes. The maximum size for GOFF objects is X'7FFFFFFF'.

DASM040S Missing operand

A required operand is not present in the statement.

DASM041E Term expected; text is unclassifiable - xxxxx

A term in an expression was expected, but was not found.

DASM042E Length attribute of symbol is unavailable; default=1- xxxxx

The length of an undefined symbol was requested, or the assembler could not determine the length during look-ahead processing. The assembler will use the value of 1.

DASM043E Previously defined symbol - xxxxx

The given symbol was previously defined in an `EXTRN` or `WXTRN` statement or in the label field of a previous statement.

DASM044E Undefined symbol - xxxxx

The given symbol was not defined in an `EXTRN`/`WXTRN` statement or in the label field of a statement.

DASM045E Register not previously used - xxxxx

The given register was specified in a `DROP` statement but was not previously used in a `USING` statement.

DASM046E Bit 7 of CCW flag byte must be zero

Bit 7 of the flag byte of a `CCW`, `CCW0` or `CCW1` statement is not zero.

DASM047E Severity code too large

The severity code specified in an `MNOTE` statement must be between 0 and 255 or the `*` character.

DASM048E ENTRY error - xxxxx

The specified symbol is used as an operand to the `ENTRY` statement, but was either undefined, or used in an incompatible fashion.

DASM050E - Illegal name field; name discarded - xxxxx

The given name is either an invalid name for a macro prototype statement or a `COPY` statement.

DASM051E - Illegal statement outside a macro definition

An `MEND`, `MEXIT`, `ASPACE`, `AEJECT` or `AREAD` statement has been encountered when not processing a macro definition.

DASM054E Illegal continuation record

The maximum number of continuation cards (10) has been encountered, or the end of the input file was encountered when a continuation card was expected.

DASM055S Recursive COPY

A nested `COPY` statement caused the assembler to attempt to copy a source already being copied.

DASM057E Undefined operation code - xxxxx

The given operation code is not an instruction and could not be processed as a macro invocation.

DASM058E Invalid relative address - xxxxx

The target of a relative instruction must be on an even boundary, so that the value can be represented in terms of half-words. Alternatively, the target was not within the control section.

DASM060S COPY code not found - xxxxx

The specified `COPY` member could not be located.

DASM061E Symbol not name of DSECT, DXD - xxxxx

The given symbol was used in a `Q`-type constant and is not defined, or is not associated with a `DSECT` or `DXD`.

DASM062E Illegal operand format - xxxxx

The given operand is not in the correct format for the statement, or exceeds the statement's limits.

DASM063E No ending apostrophe - xxxxx

The assembler expected a closing quote character, but found the indicated text instead.

DASM064S Floating point characteristic out of range

A floating point constant has a value which cannot be converted to IBM HFP format.

DASM065E Unknown type - xxxxx

An unknown type specification was used in a literal expression.

DASM066W 2-byte relocatable address constant

A Y-type, or 2-byte A-type, constant which contains a relocatable expression was discovered.

DASM067S Illegal duplication factor - xxxxx

A duplication factor was illegally applied to a constant, or the duplication factor is 0 or it is greater than X'FFFFFE'.

DASM068S Length error - xxxxx

The length modifier of a constant is incorrect, illegal, too long or otherwise out of range.

DASM069S Length of second operand must be less than length of first

For MP and DP instructions, the length of the source (second) operand must be less than the length of the target (first) operand. This message will not be generated if the length of either operand is explicitly coded as zero.

DASM070E Scale modifier error - xxxxx

The scale modifier may not be allowed for this constant, or it is out of range for the constant or the constant is a relocatable constant, or there is some other error in the scale modifier expression.

DASM071E Exponent modifier error - xxxxx

Either an exponent is not allowed for this type of constant, or the exponent modifier is out of range or too large, or the constant is a relocatable constant.

DASM072E Data item too large

Either a Y-type or other constant specifies a value which is larger than the constant can contain.

DASM073E Precision lost

A scale or length modifier on a floating point constant causes a loss of precision.

DASM074E Illegal syntax in expression - xxxxx

The expression has two terms without an intervening operation, or two operations without intervening terms, or the expression contains invalid or missing characters or delimiters, or the expression illegally uses relocatable terms.

DASM075E Arithmetic overflow

An expression value computed as the final result, or as an intermediate step, is too large.

DASM076E Statement complexity exceeded

An expression contains too many terms and/or operators, or the expression contains too many relocatable terms.

DASM077E Circular definition

A symbol's value depends, either directly or indirectly, on itself.

DASM079E Illegal PUSH-POP

Either more POP statements than corresponding PUSH statements were discovered, or the PUSH nesting limit (4) was reached.

DASM080E Statement is unresolvable

The statement cannot be resolved because it contains a complex relocatable expression, or it depends on an unresolved symbol, or the location counter has been circularly defined.

DASM081E Created SET symbol exceeds 63 characters - xxxxx

`SET` symbols are limited to 63 characters, including the leading ampersand.

DASM082E Created SET symbol is null - xxxxx

A variable substitution resulted in a SET symbol that was the empty string.

DASM083E Created SET symbol is not a valid symbol - xxxxx

A variable substitution resulted in a `SET` symbol that was not syntactically valid.

DASM084S Generated name field exceeds 63 characters; discarded - xxxxx

The name field on an instruction is limited to 63 characters.

DASM085I Generated operand field is null

A generated instruction produced an empty operand field.

DASM086S Missing MEND generated - xxxxx

The given macro definition file ends before the `MEND` statement was found.

DASM087S Generated operation code is null

The operation code field on a generated statement is empty.

DASM088E Unbalanced parentheses in macro call operand - xxxxx

An operand of a macro call statement had either too many, or too few parentheses.

DASM089E Arithmetic expression contains illegal delimiter or ends prematurely

The expression contains an illegal character, or arithmetic subscripts are used without the proper closing parentheses.

DASM090E Excess right parenthesis in macro call operand.

Too many closing parentheses were specified in an operand to a macro call statement.

DASM091E Character string exceeds maximum length; truncated to maximum

Character strings in SETC values or character operands are limited to 1024 characters.

DASM092E Substring expression 1 points past string end; default=null

The first expression in a substring operation indicates a position past the end of the string, an empty string will be used.

DASM093E Substring expression 1 less than 1; default = null

The first expression in a substring operation is 0 or negative, an empty string will be used.

DASM094I Substring goes past string end; default=remainder

The second expression in a substring operation specifies a length that goes beyond the end of the string. The valid characters from the string will be used.

DASM095W Substring expression 2 less than zero; default=null

The second expression in a substring operation specifies a negative length, an empty string will be used.

DASM096E Unsubscripted SYSLIST; default=SYSLIST(1)

The SYSLIST variable was used without being subscripted, the subscript will default to 1.

DASM097E Invalid attribute reference to SETA or SETB symbol; default=U or 0 - xxxxx

A SETA or SETB symbol was referred to with the `L'`, `S'`, `T'` or `D'` attribute.

DASM098E Attribute reference to invalid symbol; default=U or 0 - xxxxx

An `L'`, `N'`, `S'`, `D'` or `T'` attribute refers to an invalid symbol.

DASM099W Wrong type of constant for S' or I' attribute reference; default=0 - xxxxx

An `I'` or `S'` attribute reference uses a symbol which is not decimal, fixed-point or floating-point.

DASM100E Subscript less than 1; default to subscript=1 - xxxxx

A subscript in a symbol is less than 1, a subscript value of 1 will be used.

DASM102E Arithmetic term is not self-defining term; default=0 - xxxxx

An invalid self-defining term was used in an expression or a SETC term.

DASM103E Multiplication overflow; default product=1

An expression involving a multiplication produced a value which is too large, the value of 1 will be used.

DASM105U Arithmetic expression too complex

An internal work buffer has overflowed because of an expression in a macro definition statement was too large, containing too many terms and/or operations.

DASM106E Wrong target symbol type; value left unchanged - xxxxx

An attempt was made to assign to a previously defined SET symbol with the wrong type, the statement is ignored.

DASM107E Inconsistent dimension on symbol; subscript ignored or 1 used - xxxxx

A subscript was used on a SET symbol which was not declared as dimensioned, or a dimensioned SET symbol does not have a subscript, or a previously defined SET symbol's dimension doesn't match the symbol definition, or multiple dimensions were used on a symbol which was not a macro operand.

DASM109E Multiple operands for undimensioned SET symbol; gets last operand - xxxxx

An assignment was made to an undimensioned SET symbol using multiple operands, the last operand is used.

DASM110S Library macro first statement not 'MACRO' or comment

The first non-comment statement in a macro definition member must be the `MACRO` prototype statement.

DASM111S Invalid AIF or SETB operand field - xxxxx

The operand field of an AIF or SETB statement is either missing or does not begin with a left parenthesis.

DASM112S Invalid sequence symbol - xxxxx

A sequence symbol is either syntactically invalid, or appears in a name field as the result of macro substitution, or the operand of an AGO or AIF statement is blank.

DASM113S Continue column blank

The continue column in a card following a continued card was not set.

DASM114S Invalid COPY operand - xxxxx

The copy member name specified in a `COPY` statement is invalid. The first character of a `COPY` member name must be alphabetic, with the name being up to 8 characters long.

DASM115S COPY operand too long

The operand of a COPY statement must be no longer than 8 characters.

DASM116E Illegal SET symbol

A `SET` symbol, used in the name field of a `SET` instruction or in the operand field of the `GBL` or `LCL` instruction, must begin with an ampersand followed by at most 62 characters.

DASM117E Illegal subscript - xxxxx

The subscript was either an invalid expression, or has too many parentheses.

DASM118S Source macro ended by 'MEND' in COPY mode

A `COPY` file encountered in a macro definition ended in an `MEND` statement, ending the macro.

DASM119S Too few MEND statements in COPY code

A `COPY` file contains a macro definition, but the end of the `COPY` file was encountered before the appropriate `MEND` statement. An `MEND` statement will be inserted.

DASM120S EOD where continuation record expected

The end of the input file was encountered when a continuation card was expected.

DASM122S Illegal operation code format - xxxxx

The operation field of an instruction is either missing or is not followed by a blank.

DASM123S Variable symbol too long - xxxxx

Variable symbols are limited to 62 characters after the initial ampersand.

DASM124S Illegal use of parameter

A variable symbol has been used as a `SET` symbol and as a macro parameter.

DASM125S Illegal macro name - macro uncallable

The macro name specified in the statement is not valid symbol.

DASM126S Library macro name incorrect

The macro name in a macro prototype statement must be the same as the macro file name used to define the macro when it is invoked.

DASM127S Illegal use of ampersand

An ampersand was produced from a macro substitution, or a macro parameter or character constant contained an odd number of ampersands.

DASM128S Excess right parenthesis

Too many closing parentheses were discovered in a statement.

DASM129S Insufficient right parentheses - xxxxx

Too many open parentheses are present in the source.

DASM130S Illegal attribute reference - xxxxx

The symbol following an attribute reference is not a valid symbol, or the quote is missing from a `T'` attribute reference, or the `N'` attribute is only allowed on macro operands

DASM132S Invalid logical expression

An invalid logical expression was used in an `SETB` or `AIF` statement.

DASM137S Invalid character expression - xxxxx

An invalid character expression was used in a `SETC` statement, or in a character expression.

DASM138W Non-empty PUSH xxxxx *stack*

The PUSH stack described in *xxxxx* was not empty at the end of assembly. This indicates there were more PUSH instructions than POP instructions. The option *–f NOPUSH* can be used to suppress this warning.

DASM139S EOD during REPRO processing

The end of the input file was discovered immediately following a `REPRO` statement.

DASM140W END record missing

The end of the input file was discovered without the expected `END` statement.

DASM141E Bad character in operation code - xxxxxxx

The operation code for a statement must be alphanumeric, the characters A thru Z, 0 to 9, $, #, @ or _ are allowed. Spaces are not allowed.

DASM142E Operation code not complete on first record

The entire operation code for a statement must be present on the first record of the statement and cannot be continued on a continuation record.

DASM143E Bad character in name field - xxxxxx

The name field for a statement must be alphanumeric, the characters A thru Z, 0 to 9, $, #, @ or _ are allowed. Spaces are not allowed.

DASM144E Begin-to-continue columns not blank

One or more columns before the continuation column on a continuation record are not blank.

DASM145E Operator, right parenthesis, or end-of-expression expected

The assembler expected a continuation of the expression via another operator, or the end of a parenthesized expression .

DASM147E Symbol too long, or first character not a letter - xxxxx

The symbol is longer than the assembler limit of 63 character, or does not begin with a letter or the underscore character.

DASM148E Self-defining term lacks ending quote or has bad character - xxxxx

A binary (`B'`) or hexadecimal (`X'`) self-defining term contains invalid characters or is missing the closing quote.

DASM149E Literal length exceeds 256 characters, including = sign - xxxxx

The literal is longer than 256 characters.

DASM151E Literal expression modifiers must be absolute and predefined

The length modifier or duplication factor is not an absolute expression, a self-defining expression or uses an undefined symbol.

DASM152S External symbol too long or unacceptable character - xxxxx

The external symbol exceeded the assembler length or contains an invalid character.

DASM153S START statement illegal - CSECT already begun

A `START` statement was found after the beginning of a control section.

DASM154E Operand must be absolute, predefined symbols; set to zero

The expression used in an operand for this statement either uses undefined symbols, or is not an absolute value. Zero will be substituted.

DASM155S Previous use of symbol is not this section type - xxxxx

This symbol was used previously and then encountered on a section defining statement.

DASM156S Only ordinary symbols, separated by commas, allowed - xxxxx

Only a properly defined symbol is allowed as the operand in this statement.

DASM157S Operand must be a simply-relocatable expression

The operand is not properly defined or is not a simple relocatable expression.

DASM159S Operand must be absolute, proper multiples of 2 or 4

The operands in a `CNOP` statement were not multiples of 2 or 4 or were not absolute, defined expressions.

DASM160W Invalid BYTE function operand - xxxxx

The value of the operand of a `BYTE` built-in function must be in the range 0-255.

DASM161W Only one TITLE statement may have a name field

The assembler discovered a second `TITLE` statement with a name field; only the first `TITLE` statement may have a name field.

DASM162S PUNCH operand exceeds 80 columns; ignored

A `PUNCH` statement attempted to write more than 80 characters on a single record. The `PUNCH` statement is ignored.

DASM163W Operand not properly enclosed in quotes

The assembler expected the operand of a `PUNCH`, `TITLE`, or `MNOTE` statement to be enclosed in quotes.

DASM164W Operand is null string - record not punched

The operand of a PUNCH statement is the empty string, the PUNCH statement is ignored.

DASM165W Unexpected name field - xxxxx

The assembler expected the name field on this statement to be blank or a sequence symbol.

DASM167E Required name missing

The statement requires a value in the name field but is blank or contains a sequence symbol.

DASM169I Implicit length of symbol symbol *used for operand* - n

An explicit length was not specified in an SS-format instruction. The assembler used the implicit length implied by the operand. This message is only produced if the IMPLEN flag is specified.

DASM170S Error logging capacity exceeded

The internal buffer area used between assembler phases to log messages was exceeded.

DASM171S Standard value too long - xxxxx

The default value for a macro operand was longer than the limit of 255 characters.

DASM172E Negative duplication factor; default=1 - xxxxx

The duplication factor on the statement is negative, the assembler has substituted a duplication factor of 1.

DASM173S Delimiter error, expected blank - xxxxx

The assembler expected a blank character but encountered some other text.

DASM174S Delimiter error, expected blank or comma - xxxxx

The assembler expected a blank or comma character but encountered some other text.

DASM175S Delimiter error, expected comma - xxxxx

The assembler expected a comma character but encountered some other text.

DASM178S Delimiter error, expected comma or right parenthesis - xxxxx

The assembler expected a comma or right parenthesis character but encountered some other text.

DASM179S Delimiter error, expected right parenthesis - xxxxx

The assembler expected a right parenthesis character but encountered some other text.

DASM180S Operand must be absolute

The first, third or fourth operand of a `CCW` instruction must be an absolute value.

DASM181S CCW operand value is outside allowable range

The first operand of a `CCW` instruction must be between 0 and 255. The second operand must be between 0 and 16,777,215 for the `CCW` and `CCW0` instructions, or between 0 and 2,147,483,647 for the `CCW1` instruction. The third operand must be between 0 and 255 and be a multiple of 8. The fourth operand must be between 0 and 65,535.

DASM182E Operand 2 must be absolute, 0-65535; ignored

The second operand on this statement must be a defined, absolute value in the range 0 to 65,535.

DASM183E Operand 3 must be absolute, 0-255; ignored

The third operand on this statement must be a defined, absolute value in the range 0 to 255.

DASM186E AMODE/RMODE already set for this ESD item

A redundant `AMODE/RMODE` instruction was discovered for the ESD in the name field of the instruction.

DASM187E The name field is invalid - xxxxx

The name field for the instruction is not a valid control section, an `ENTRY` name or a valid external name.

DASM188E Incompatible AMODE and RMODE attributes

A symbol cannot be both `AMODE 24` and `RMODE ANY`.

DASM192W Lost precision - underflow to zero

The value is too small to be represented, a value of 0.0 will be used.

DASM193W Lost precision - underflow to denormal

The value is too small to be represented in normalized form, but can be represented in denormalized form.

DASM198E Exponent modifier is not permitted for special value

A floating-point special value may not have an exponent modifier.

DASM199E Rounding indicated invalid

An invalid value was used as the rounding indicator for a floating-point constant.

DASM212W Branch address alignment unfavorable

The target of a branch instruction was not properly aligned.

DASM213W Storage alignment unfavorable

An address operand of the instruction is not aligned as the instruction requires.

DASM214E Invalid operand value

The value of an operand was not correct for the operation or assembler function. Or, the operand was too long to be processed.

DASM216W Quad-word alignment in NOGOFF object text

The SECTALGN value specifies that sections are to be aligned on a quadword (16 byte) boundary in an old-style object.

Newer IBM linkers will support this alignment, but older ones may not, check with the linker documentation to ensure that this usage is supported.

DASM253C Too many errors

The number of errors for a statement has been exceeded.

*DASM254I *** MNOTE ****

An `MNOTE` statement generated message.

DASM303W Multiple address resolutions may result from this USING and the USING on statement number

The base address specified by the `USING` statement overlaps with the base address from previous `USING` at the specified statement number.

Because of this overlap, the assembler may choose one or the other specified base register, which may not be the programmer's intent.

This is frequently caused by the inadvertent lack of a `DROP` statement.

This message can be suppressed via the `USING(WARN(n))` option, specifying a value of less than '4' for 'n'.

DASM305E Operand 1 does not refer to location within reference control section

An incorrect value was specified as the first operand for a dependent `USING` statement. The `USING` statement is ignored.

DASM307E No active USING for operand n

The given operand, *n*, was used but the assembler was not able to determine an active USING specification to provide a base register.

DASM309W Operand resolved to a displacement with no base register

An operand of a machine instruction refers to a memory location, but register 0 was specified as the base register. Thus, a value of zero will be used and only the displacement portion of the address is meaningful.

This will access low-address memory, and may be what the programmer intended but is probably a program error.

The *–NOPAGE0* flag can be used to disable this warning.

DASM310W Name already used in prior ALIAS or XATTR - xxxxx

The specified name was previously used in an `ALIAS` or `XATTR` statement.

DASM311E Illegal ALIAS string - xxxxx

An `ALIAS` string must be a non-null constant of the form C'ccccc' or X'hhhhhh' and must contain characters in the range X'42' to X'FE'.

DASM312E ALIAS name is not declared as an external symbol - xxxxx

The specified name does not appear in an `EXTRN` or `CSECT` statement, or is not implicitly declared as an external symbol via a V-type constant.

DASM315E XATTR instruction invalid when NOGOFF specified

The `XATTR` instruction is not valid if the output object format is not GOFF. The `XATTR` statement is ignored.

DASM320W Immediate field operand may have incorrect sign or magnitude

The specified immediate operand is outside of the range specified for the instruction. The assembler continues processing using only the number of bits specified in the instruction format.

To disable this warning, use the *–xtc* option, or the `NOTYPECHECK` specification in an `ACONTROL` statement.

DASM400N Error in invocation parameter - xxxxx

The command-line parameter specified in *xxxxx* was either unrecognized or incorrectly specified. The text *xxxxx* will contain more information.

*DASM420N Error in a *PROCESS statement parameter* - xxxxx

The assembler was unable to process an option or a sub-option of a *PROCESS parameter, due to either the sub-option being invalid, or a mis-specification.

*DASM422N Option xxxxxxxx is not valid in a *PROCESS statement*

The specified option was not recognized, or can not be used in a *PROCESS statement.

DASM430W - Continuation statement does not start in continue column.

The operand field on a continued statement ends with a comma, indicating that the statement is continued on the following statement. However, a blank was found in the continuation column on the following statement.

The continuation is ignored.

DASM431W - Continuation statement may be in error - continuation indicator column is blank

A list of oeprands ends in a command, but the continuation indicator column (by default, column 72) is blank.

The continuation is ignored.

DASM432W - Continuation statement may be in error - comma omitted from continued statement

A continued statement starts in the continue column (by default, column 16), but the previous statement did not have a trailing comma in its operands.

Continuations after this statement are ignored.

DASM433W - Statement not continued - continuation statement may be in error

A continued statement is full, but the continuation does not start in the continue column (by default, column 16.)

Continuations after this statement are ignored.

DASM435I - Record n *in* xxxxxxx

When the *–fmesg=dasmx* or *–fmesg=gccx* option is specified, the first form of this this message is produced, and contains the original text of the lines which produce any subseuqent diagnositc message(s).

When the *–f RECORD* option is enabled, the second form of this message is generated following any diagnostics, and contains the original line number and file name of the statement that caused the preceding diagnostic messages.

Because *–fmesg=dasmx* or *–fmesg=gccx* and *–f RECORD* may both be simulaneously enabled, it's possible for both forms of this message to be produced by any diagnostics; one preceeding the diagnostic messages, and one following.

DASM500W Requested alignment exceeds section alignment

The section alignment, specified via the *–sectalgn* option, is smaller than the alignment needed in the instruction, typically for a literal value. The needed alignment may not be honored.

DASM900W Input line too long, truncated

An input line was longer than the assembler limit of 80 characters, it has been truncated to 80 characters. This message can be disabled with the *–flonglines* option.

DASM901E Scale modifier is not permitted for special value

A "special value", for floating point or decimal constants, was used in the source with a scaling value, which is not allowed. Remove the scale specification.

DASM902E Invalid floating point special value - xxxxx

The floating point special value did not have a closing parenthesis, or was not one of the floating-point special values.

DASM903W ALIAS name is not declared prior to setting flag

An `ALIAS MAPPED` or `ALIAS FUNCTION` statement was encountered prior to defining the ALIAS name.

DASM909W G-type constant not supported

DASM does not support G-type constants. This message may be removed in a future release.

DASM911E Concatenation character not followed by apostrophe

A '`.`' concatenation was specified, but the item being concatenated is not a string function or a string constant.

DASM913W RMODE and AMODE have no effect in Linux mode

When outputting Linux ELF objects, `RMODE` and `AMODE` opcodes have no effect. The mode is determined entirely by the *–flinux* or *–flinux64* options. Note that Linux/390 cannot be mixed with z/Linux code without extreme care.

DASM914S Illegal address reference length

An attempt was made to reference data such as using a `A`-type constant that had an unusual size not supported by ELF. For example, a `DC AL3(xxx)` opcode would cause this error. ELF only allows 1, 2, 4, or 8-byte relocations.

License Information File

DASM consults the license information file each time it is executed. The file includes the licensee name, expiration date, license key, and other pertinent information.

This file must be accessible or the assembler will not execute.

On UNIX and Windows host platforms, the file is named `dignus.inf` and is found in the same directory as the **dasm** executable. The `dignus.inf` file is a text file which can be edited by any text editor. However, changing the expiration date, licensee, options or host platform definitions will invalidate your license.

On z/OS, the license information file is named `DIGNUS` and is found in the same load module PDS as the **DASM** executable. `DIGNUS` is in load module format, and is generated from assembly language source. To make changes in the license information, the assembly language source must be changed, assembled and link-edited to produce the `DIGNUS` load module. However, changing the expiration date, licensee, options or host platform definitions will invalidate your license.

Your `dignus.inf`, or `dignus.asm` assembly language source to create `DIGNUS,` is provided separately from the installation materials. Editing this file is part of the installation process, and is described further there.

If you have licensed other products from Dignus, LLC, the license information can simply be concatenated with the other information into one `dignus.inf` or `dignus.asm` file.

ASCII/EBCDIC Translation Table

DASM uses the following tables to translate characters between ASCII and EBCDIC. These tables represent the traditional mapping of the IBM Code Page 1047 (IBM1047) to ISO LATIN-1.

However, this is not the official IBM1047 mapping. The official mapping maps EBCDIC X'15' to LINEFEED X'85' and maps EBCDIC X'25' to NEWLINE X'0A'. This is reversed from their traditional mappings. **DASM** uses the traditional map.

The —*tr* option can be used adjust this mapping.

ASCII to EBCDIC

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8	20	21	22	23	24	25	06	17	28	29	2A	2B	2C	09	0A	1B
9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A	41	AA	4A	B1	9F	B2	6A	B5	BB	B4	9A	8A	B0	CA	AF	BC
B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
D	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	BA	AE	59
E	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
F	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

EBCDIC to ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	0A	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	85	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	A2	2E	3C	28	2B	7C
5	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	21	24	2A	29	3B	5E
6	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	A6	2C	25	5F	3E	3F
7	F8	C9	CA	CB	C8	CD	CE	CF	CC	60	3A	23	40	27	3D	22
8	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
A	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	5B	DE	AE
B	AC	A3	A5	B7	A9	A7	B6	BC	BD	BE	DD	A8	AF	5D	B4	D7
C	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
E	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
F	30	31	32	33	34	35	36	37	38	39	B3	DB	DC	D9	DA	9F