

**How HLASM Can Help You Find and Fix
Assembler Language Problems**

SHARE 120 in San Francisco, Session 12673

John R. Ehrman
ehrman@us.ibm.com

IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, California 95141 USA

February 2013

Overview: How HLASM Can Help

- Things HLASM can help with:
 - Information available in the listing
 - The program being assembled
 - The assembly environment
 - How to reveal possibly-hidden information
 - Useful options
 - Optional diagnostics
 - Macro-related information and problem solving
 - Other things worth noting
- Things HLASM can't help with: (Sorry!)
 - Problems with program structure, logic, or style
 - HLASM Toolkit components can help with these
 - Especially the Program Understanding Tool and the Structured Programming Macros!
 - Other useful components: Source cross-reference utility, File comparison utility, Symbolic debugger, Disassembler
 - Problems with using the wrong files (such as libraries)
 - Resource constraints (but HLASM can often cope)

Information in the Listing

- Options Summary
- Assembler Service Status (INFO option)
- External Symbol Dictionary (ESD option)
- Source and Object Code
 - Active-USINGs Heading
- Relocation Dictionary (RLD option)
- Ordinary Symbol and Literal XREF (XREF option)
 - Unreferenced Symbols in CSECTs
- Macro and COPY Code Summary (MXREF option)
 - Macro and COPY Code XREF
- DSECT XREF (DXREF option)
- USING Map (USING(MAP) option)
- General Register XREF (RXREF option)
- Diagnostic XREF and Assembler Summary

Options Summary

- Listing shows options in effect, and options hierarchy for overrides

```
Overriding ASMAOPT Parameters – NODXREF,NODECK      ← ASMAOPT file
Overriding Parameters– asa,noobj,exit(prtexit(prtx)) ← Invocation PARMs
Process Statements–  OVERRIDE(CODEPAGE(X'047B'))     ← *PROCESS
                   NOESD                           ← *PROCESS
```

Options for this Assembly

```
NOADATA
  ALIGN
3  ASA
  BATCH
1  CODEPAGE(047B)
  NOCOMPAT
  NODBCS
2  NODECK
2  NODXREF
5  NOESD
3  EXIT(PRTEXTIT(PRTX))
  - - -  etc.
```

- Numeric tags in left margin indicate the origin of the override
- **Check:** correct options; exits; BATCH; APAR status (listing line 1)

Assembler Service Status (INFO Option)

- HLASM prints its service status, other useful information
 - Latest PTF number is *always* on the *first* line of the listing
- Example of the printed text:

The following information describes enhancements and changes to the High Level Assembler Product.

The information displayed can be managed by using the following options:

INFO – prints all available information for this release.

INFO(yyyymmdd) – suppresses items dated prior to "yyyymmdd".

NOINFO – suppresses the product information entirely.

20100204 APAR PM06119 Fixed
NOCOMPAT option not recognized on invocation parms
when COMPAT(SYSLIST) has been set as the default in
ASMADOPT.

20100316 APAR PM09235 Fixed
Alignment for XD item in ESD incorrect and shows doubleword
alignment instead of the correct alignment for the length

- **Check:** current service status; language changes

External Symbol Dictionary (ESD Option)

- The external names defined and referenced by this assembly
- Standard assembly

Symbol	Type	ID	Address	Length	Owner Id	Flags	Alias-of
SECT_A	SD	00000001	00000000	0000002C		01	
MYCOM	CM	00000002	00000000	00000060		00	
MY_XD	XD	00000003	00000007	00000018			
SECT_B	SD	00000004	00000030	00000038		02	
BDATA	LD		00000030		00000004		

- Quadword-aligned sections, SECTALGN(16) option (more at slide 24)

Symbol	Type	Id	Address	Length	Owner Id	Flags	Alias-of
SECT_A	SQ	00000001	00000000	00000026		00	
MYCOM	CQ	00000002	00000000	00000060		00	
MY_XD	XD	00000003	0000000F	00000018			
SECT_B	SQ	00000004	00000000	00000038		00	
BDATA	LD		00000030		00000004		

- Names are normally in upper-case letters

External Symbol Dictionary (ESD Option) ...

- Each item (except LDs) is assumed to be independently relocatable
- Each symbol has a type and an identifying number (its “ESD ID”)
 - Section definitions (types SD, CM, PC; or SQ, CQ, PQ if quad aligned)
 - PC sections may cause MODE problems, even if zero length
 - Usual cause: EQUs appearing before first section is initiated
 - Entry point definitions (type LD)
 - LD-ID identifies the section in which the symbol is an entry
 - External references (types ER, WX)
 - Names of symbols referenced by this assembly but defined elsewhere
 - External Dummy definitions (type XD)
 - Symbols naming DXD instructions, or DSECT names in Q-cons
- ALIAS information
 - ALIAS instruction changes an existing external name to another
 - Linkers and loaders see the changed name, not the original
- **Check:** correct name/length/type; mixed-case aliases; Private Code (blank section name); AMODE/RMODE; NOTHREAD option (each subsequent CSECT starts at 0)

Source Program and Object Code Listing

- Source and object code listing
 - Active USINGs heading lines
 - LOC, C-LOC, D-LOC, R-LOC location counter headings
 - Indicates type of section active at start of the page
 - USING resolution details: registers, offsets
 - Statement-origin prefix characters
- Statements and options can affect the source and object code listing
 - PRINT instructions control various portions of the listing
 - PCONTROL option can override PRINT-instruction controls (see slide 20)
 - USING and FLAG options control various diagnostics (see slides 20, 32)
- To suppress the source and object code listing
 - Selectively: use PRINT statement operands (see slide 19)
 - Completely: use NOLIST option (but it suppresses the entire listing!)
- **Check:** code is in correct sections; END-nominated execution entry

Diagnostic Messages and Severities

- All messages prefixed with '** ASMA'
- Final letter of ASMAⁿⁿⁿX is a severity indicator:

Letter	Severity	Meaning
I	0	Information
N	2	Notification
W	4	Warning
E	8	Error
S	12	Severe Error
C	16	Critical Error
U	20	Unable to proceed

- If FLAG(RECORD) is specified, all messages are followed by another indicating the source record to which the message applies
 - Also identifies records from macro and COPY-file data sets
- ** ASMA435I Record 26 in TATST ASSEMBLE A1 on volume: EHR191**
- SUPRWARN option can suppress specified low-severity messages (slide 23)

Relocation Dictionary (RLD Option)

- Information about relocatable (and Q, CXD) **address constants**
- **Position ID**: ESDID of the section where the constant resides
- **Relocation ID**: ESDID of the name whose value the adcon will contain
- **Address**: the address or offset at which the constant resides within its section, as specified by the Position ID
- **Length**: the length of the adcon
- **Type**: the type of the adcon
- **Action**: Store (ST) or add/subtract (\pm) the relocation value
- **Check**: intended relocatable items; overlapping RLDs; complexly relocatable operands

Ordinary Symbol and Literal Cross-Reference (XREF Option)

- XREF has three sub-options (default: SHORT, UNREFS)
 - XREF(FULL) for all symbols, referenced or not
 - XREF(SHORT) for referenced symbols only
 - XREF(UNREFS) lists unreferenced non-DSECT symbols
 - Helps locate dead code, dead data items
 - Ignored if XREF(FULL) is specified
- Displays information about each symbol (example on next slide)
 - Symbol, length attribute, value
 - Relocation ID, relocatability tags (especially “C”), symbol type, where defined
 - References, including tags indicating use: (see next slide)
Bbranch, Drop, Modification, Using, eXecute
 - Symbol Batch_Init is a branch target (B tag)
 - Symbol Err_Buff modified (M tag); a USING base (U tag)
 - Symbol Move_Msg is executed (X tag)
 - Symbol R1 appears in USING (U tag) and DROP (D tag) instructions
- **Check:** usage tags; relocation ID and type; attributes; duplicate or replaceable literals

Ordinary Symbol and Literal Cross-Reference Example

Symbol	Length	Value	Id	R	Type	Asm	Program	Defn	References
...									
BadEQU	1	000004	0001	C	U			666	← note C relocation
Batch_Init	2	00035C	0001		H			493 399B 490B	← note B tag
Batch_Len	8	000018	FFFF	A	U			772 497	
Batch_Msg_1	39	0006A1	0001		C			681 469 469 716	
...									
Err_Buff	1	000000	FFFD		J			787 127U 517M	← note U tag
Err_Msg	255	000000	FFFD		C			788 277M 481M 486M	← note M tags
ESD_Item	1	000000	FFFB		J			804 349U 831	← note U tag
...									
Move_Msg	6	0004A6	0001		I			645 641X	← note X tag
PPtr	QUALIFIER								
...									
...									
R1	1	000001	0001	A	U	GR32		53 123U 128D 132M 133M 133 135 215D 248M 262 348M 349U 373M	note U,D tags
...									
...									
...									

- B=Branch target, D=Drop, M=Modified, U=Using, X=Executed

Macro/COPY Summary and Cross-Reference (MXREF Option)

- MXREF option has three sub-options:
 - MXREF(SOURCE) (default option) shows where each macro/COPY originated
 - MXREF(XREF) shows where each macro/COPY is referenced
 - MXREF(FULL) is equivalent to MXREF(SOURCE,XREF)
- Macro/COPY usage information
 - Information about library data sets and members
 - COPY and LIBMAC tags, where defined, who called
 - Inner macro calls captured, even if not in the listing
 - Inner macro calls and COPY statements aren't visible to source-cross-reference tools!
 - COPY-reference statement numbers tagged with **C**
- MXREF data also written to SYSADATA file
 - ASMAXADT sample ADATA exit summarizes “Bill of Materials” info
- **Check:** files from correct libraries; inner macro's callers; duplicate COPY; missing macro names

DSECT Cross-Reference (DXREF Option)

- DXREF option lists all DSECTs and DXDs defined in the assembly
 - Displays name, length, relocation ID, definition-start statement number
 - Relocation ID: Identifies the dummy section in which each symbol is defined
 - Starts at X'FFFFFFFF' for DSECTs, counts down
 - Starts at X'00000001' for external dummy symbols, counts up (same as for ESDIDs)

- Example:

Dsect	Length	ID	Defn
AEFNPARM	0000001C	FFFFFFFF	165 (negative ID for internal dummy section)
AEFNRL	00000024	FFFFFFFE	183
B	00000008	00000002	42 (positive ID for external dummy section)

- **Check:** DSECTs are intended; correct DSECT and DXD lengths

USING Map (USING(MAP) Option)

- USING Map summarizes all USING/DROP activity
- Statement-location data
 - Statement number of the USING or DROP
 - Active Location Counter and section ID in which the statement appeared
- The type of action specified (USING, DROP)
- Type of USING (Ordinary, Labeled, Dependent, Labeled Dependent)
- Base address, range and Relocation ID of each USING
- Anchoring register(s) on which the USING instruction is based
- Maximum displacement and last statement resolved based on this USING
 - Helps you to minimize USING ranges, avoid unwanted resolutions
- The operand-field text of the USING statement
- **Check:** max displacement; last resolved statement; un-DROPped regs

General Register Cross-Reference (RXREF Option)

- Implicit references noted (e.g., statement 116: STM instruction)

LM 3,5,X implicitly references (and modifies) GR 4

- Actual register use; does not depend on symbolic register naming!

Register References (M=modified, B=branch, U=USING, D=DROP, N=index)

0(0)	116	163M	164	179M	180	181	185M	186M	186	190	...
	374M	388M	389M	389	450M	456M	473M	474M	475	477M	...
	... etc.										
2(2)	116	171M	174M	197M	198M	199	295M	357M	358M	359	...
	419M	420	421M	422N	528M	568M	625M	625	626M	627	...
	... etc.										
12(C)	116	117M	119U	295M	528M	568M	649D				...
13(D)	116	178	180	181M	293M	295	309	311	312M	524M	...
14(E)	116	295M	296B	399M	490M	498B	528M	529B	568M	569B	...
15(F)	109U	111	116	117	118D	189M	190	295M	528M	568M	...

- Register 2 used as index at statement 422 (N tag)
- Register 14 used in branch statements (296, 498, etc.; B tag)
- Registers used for base resolution not referenced or tagged
- **Check:** low utilization; localized loads/stores; based branches

Assembly Summary

Last part of the listing:

- A compact diagnostic summary: statement, origin, severity
 - Pointers to origins of source statements having diagnostics
 - Format is `sn(sc[:mac],nnn)`, where
sn = statement number; s = Pprimary/Library, c = concatenation number,
mac = macro name, nnn = record number in that file
- All files/data sets allocated
- Assembler and host system data
- Summary of suppressed warning messages (severity ≤ 4)
- External function statistics
- I/O exit statistics
- Storage utilization data, I/O record counts
- Assembly start/stop and processor time
- **Check:** I/O exits; I/O counts; correct library file ordering; suppressed-messages and message counts; memory use; CPU time

Assembler Options and Diagnostics

- TERM: strongly recommended; always displays a one-line summary
 - Messages displayed (if not suppressed by FLAG or SUPRWARN options) whether or not PRINT-suppressed in the listing
- BATCH: multiple assemblies with one HLASM invocation
 - Note possible “module contamination”
 - Detailed examples in the text handout
- PCONTROL: many suboptions (see slide 20)
 - Useful for “exposing” hidden listing information
- SUPRWARN: suppress chosen low-severity messages (see slide 23)
- SECTALGN: section alignment (see slide 24)

Assembler Options and Diagnostics ...

- TRANSLATE: character constants (see slide 24)
 - TRANSLATE and COMPAT(TRANSDT): translate character self-defining terms
- FLAG: controls various useful diagnostics (see slide 25)
- USING: controls several diagnostics, USING Map (see slide 32)
- TYPECHECK: validates instruction register and immediate operands
- LANGUAGE: select national language for messages, headings
- LIST(133): Wider listing provides more detail
- The ACONTROL statement provides dynamic control of many options (see slide 46)
- **Check:** TERM option; BATCH option (dangling statements, multiple assemblies)

PRINT Instruction Operands

- PRINT instruction operands affect the source and object code listing
 - **ON, OFF:** control display of source/object code listing
 - Be careful: PRINT OFF also disables message display on the listing
 - Messages are always visible if TERM option is specified
 - **DATA, NODATA:** control display of DC-generated data
 - **GEN, NOGEN:** control display of conditional-assembly generated statements
 - **MCALL, NOMCALL:** control display of inner macro calls
 - **MSOURCE, NOMSOURCE:** control display of macro-generated source statements
 - **UHEAD, NOUHEAD:** control display of Active-USINGs heading
- NOPRINT operand allowed on PRINT, PUSH, POP
 - Allows these statements to hide themselves!
 - But PCONTROL option can override them (see next slide)

PCONTROL Option

- PCONTROL lets you override PRINT operands without source changes
 - You can see full details that might have been hidden
- Sub-options are the same as PRINT instruction operands!
(Compare slide 19)
 - ON, OFF (ON exposes everything hidden by PRINT OFF statements)
 - DATA, NODATA
 - GEN, NOGEN (GEN exposes everything hidden by PRINT NOGEN statements)
 - MCALL, NOMCALL (MCALL exposes all inner macro calls)
 - MSOURCE, NOMSOURCE
 - UHEAD, NOUHEAD
- GEN, MCALL, MSOURCE are useful for macro-related problems

PCONTROL(MCALL) Option

- Controls display of inner macro calls
- Suppose you write these three simple macros:

<pre>Macro TOP &a,&b,&c MIDDLE &c,&a,&b MEnd</pre>	&n	<pre>Macro MIDDLE &x,&y,&z SetA &x*&y+3*&z BOTTOM &n MEnd</pre>	<pre>Macro BOTTOM &j MNote '&j' MEnd</pre>
---	----	---	---

- The TOP macro transposed its arguments in the call to MIDDLE; if invoked with NOMCALL active, no inner calls are visible:

```
*Process PControl (NoMCALL)
      TOP    2,3,5      Expect: 2×3+3×5 = 21
+19                                     Hmm... What went wrong ??
```

- When TOP is called with MCALL active, inner calls are visible:

```
*Process PControl (MCALL)
      TOP    2,3,5      Expect: 2×3+3×5 = 21
+      MIDDLE 5,2,3      Aha! Arguments were rearranged!
+      BOTTOM 19         Got 5×2+3×3 = 19 instead
+19
```

PCONTROL(MSOURCE) Option

- Controls display of source statements generated by macro expansions
- Expansion with MSOURCE displays all generated statements

				12	MVC2	Buffer,=C'Message'		
000000	D500	0000	C090	00000	00090	13+	CLC	0(0,0),=C'Message'
000006				00006	00000	14+	Org	*-6
000000	4100	C006			00006	15+	LA	0,Buffer(0)
000004				00004	00000	16+	Org	*-4
000000	D206					17+	DC	AL1(X'D2',L'=C'Message'-1)
000002				00002	00006	18+	Org	*+4

- Expansion with NOMSOURCE hides the macro's inner workings

				12	MVC2	Buffer,=C'Message'		
000000	D500	0000	C090	00000	00090	13+		
000006				00006	00000	14+		
000000	4100	C006			00006	15+		
000004				00004	00000	16+		
000000	D206					17+		
000002				00002	00006	18+		

- Unlike PRINT NOGEN, you can still see the object code

Assembler Diagnostics: SUPRWARN Option

- SUPRWARN(n1,n2,...) suppresses messages with severity ≤ 4 , and the message's effect on the assembly return code
- Without message suppression (Return Code 4):

```

                R:F  00000                2      Using *,15
                R:D  00000                3      Using *,13
** ASMA300W USING overridden by a prior active USING on statement number 2
000000 40E0 F00D                0000D    4      STH   14,X+1
** ASMA033I Storage alignment for X+1 unfavorable
000004 4400 F010                00010    5      EX    0,=X'0700'
** ASMA016W Literal used as the target of EX instruction
00000C                                7 X    DS    F

```

- With message suppression (Return Code 0):

```

                R:F  00000                3      Using *,15
                R:D  00000                4      Using *,13          (No warning!)
000000 40E0 F00D                0000D    5      STH   14,X+1          (No warning!)
000004 4400 F010                00010    6      EX    0,=X'0700'      (No warning!)
00000C                                9 X    DS    F
1 *Process SUPRWARN(300,33,16)

```

- Be *very* judicious! You may hide something important!

Assembler Options SECTALGN and TRANSLATE

- SECTALGN(nnn) controls Assembler's alignment of control sections
 - nnn a power of 2. For OBJ: 8 or 16; for GOFF: $8 \leq nnn \leq 4096$
 - Avoids need for creating binder statements with PUNCH/REPRO, e.g.

```
PUNCH ' PAGE MAIN'  
PUNCH ' ORDER MAIN,SUB(P)'
```

- TRANSLATE(xx) controls translation of C-type character constants
 - Affects **all** C-type constants, and character terms if option COMPAT(TRANSDT) is also specified
 - CA, CE constant subtypes provide localized control

```
DC CA'Text' Always generates ASCII (Never translated)  
DC CE'Text' Always generates EBCDIC (Never translated)  
DC C'Text' Affected by TRANSLATE option  
LA 0,C'$' Affected by TRANSLATE and COMPAT(TRANSDT) options
```

- Can help avoid unexpected translations if TRANSLATE option is specified
- **Check:** Section alignments; presence of PUNCH/REPRO statements; use of TRANSLATE option

Assembler Diagnostics: FLAG Options

- FLAG(severity) controls which messages are displayed in the listing
- FLAG(ALIGN) controls checks for normal operand alignment
- FLAG(CONT) controls checks for common continuation errors
- FLAG(IMPLEN) checks for implicit length use in SS-type instructions
- FLAG(PAGE0) checks for inadvertent low-storage references resolved with base register zero
- FLAG(PUSH) checks at END for non-empty PUSH stack
- FLAG(RECORD) indicates the specific record in error
- FLAG(SUBSTR) checks for improper conditional-assembly substrings
- FLAG(USING0) notes possible conflicts with the implicit USING 0,0
- **Check:** ALIGN messages; continuations; implicit lengths; page-zero references

FLAG(CONT) Option

- FLAG(CONT) controls checks for common continuation errors

```
      ABCDE ARG=XYZ,      Continued macro operands      X
              RESULT=JKL  Continuation starts in column 17! ← Error
** ASMA430W Continuation statement does not start in continue column.
```

- Not all diagnosed situations are truly errors; *but* check carefully!

```
      IF      (X) Then do this or that
      DO      (This,OR,That)
      ELSE    Otherwise, do that and this      ← note comma!
** ASMA431W Continuation statement may be in error –
      continuation indicator column is blank.
```

```
      IF      (X) Then do this or that
      DO      (This,OR,That)
      ELSE    Otherwise do that and this      ← note no comma!
```

- Recommend running with continuation checking enabled
 - Control scope of checking with ACONTROL statements (see slide 46)

FLAG(IMPLEN) Option

- **FLAG(IMPLEN)** option flags use of implied length in SS-type ops
 - Target-operand length may be too short or too long:

```

                                A    DS    CL99    Wrong # bytes moved?
0000A4 D262 F063 F732 ...      MVC    A,=C'Message'
** ASMA169I Implicit length of symbol A used for operand 1
```

- Length attribute of A+1 is that of A, but 1+A's is 1:

```

                                B    EQU    *
                                DS    CL99
0000C6 D262 F064 F000 ...      MVC    A+1,B    Moves L'A bytes
** ASMA169I Implicit length of symbol A+1 used for operand 1
0000CC D200 F064 F000 ...      MVC    1+A,B    Moves one byte
** ASMA169I Implicit length of symbol 1+A used for operand 1
000000 D200 F006 F006 ...      MVC    B,A
** ASMA169I Implicit length of symbol B used for operand 1
```

- Using implicit lengths **is** a good thing! But ... use them carefully
- **Check:** instruction length fields are assembled correctly

FLAG(PAGE0) Option

- **FLAG(PAGE0)** option flags “baseless” resolutions referencing Page 0 (potentially **very** important in Access Register mode!)

```
*!      BR   14      was intended...
        B   R14     Branch to address 14
** ASMA309W Operand R14 resolved to a displacement with no base register

*!      MVC  A,=C'A'  was intended...
        MVC  A,C'A'  Move bytes to A, starting at address 193
** ASMA309W Operand C'A' resolved to a displacement with no base register

*!      LA   0,8     was intended...
        LH   0,8     (What if the 2 bytes at address 8 contained 8!)
** ASMA309W Operand 8 resolved to a displacement with no base register

*!      MVC  6(,2),B  was intended...
        MVC  6(2),B  Length 2, base register zero (protection exception?)
** ASMA309W Operand 6(2) resolved to a displacement with no base register

        L    1,0(2)  Possible AR-mode problem?
** ASMA309W Operand 0(2) resolved to a displacement with no base register
   (Generated instruction 58120000 has base register 0: no AR qualification)
```

FLAG(PUSH) Option

- Operands of PUSH and POP: USING, ACONTROL, PRINT
- Non-empty PUSH stack detected at end of assembly
 - May have incorrect USING resolutions if PUSH-USINGs don't match POP-USINGs
 - Non-empty PUSH-USING stack may be serious; PUSH-PRINT isn't
- PUSH-USING stack level shown in USING subheading:

Active Usings (1): ...etc... (follows TITLE line)

“(1)” indicates USING Push depth = 1

- **Check:** non-empty PUSH USING stack at END

FLAG(USING0) Option

- Helps catch accidental use of absolute base addresses that overlap the assembler's implicit USING 0,0

```

                A      Equ   12
                USING A,12 ← ?
** ASMA306W USING range overlaps implicit USING 0,0

4110 000A      LA      1,10
4110 C008      LA      1,20  Does R12 contain 12?
```

- Note the different resolutions: one based on register 0, one on 12

```

                USING -1000,12
** ASMA306W USING range overlaps implicit USING 0,0
4110 C3DE      LA      1,-10
4120 C3F2      LA      2,10
```

```

                USING +1000,11
** ASMA306W USING range overlaps implicit USING 0,0
4130 B3E9      LA      3,2001
4145 B0C8      LA      4,1200(5)
```

- Message ASMA306W is controlled with the FLAG(USING0) option

USING Diagnostic Messages

- Messages controlled by the USING(WARN(nn)) option (see slide 32)
 - ASMA300W, ASMA301W
Nullification of one USING by another
 - ASMA302W
Base register 0 specified with nonzero base address
 - ASMA303W
Multiple valid resolutions
 - ASMA304W
Resolved displacement exceeds the limit you specified

We'll see examples on slides 33, 34, 37

- Message not controlled by an option:
 - ASMA308W
Repeated register in USING
- Message controlled by the FLAG(USING0) option (see slide 30):
 - ASMA306W
USING range overlaps implicit USING 0,0
- **Check:** examine all USING-related messages carefully

Assembler Diagnostics: USING Option

The USING option supports three sub-options:

1. MAP: controls Using Map in the listing (see slide 14)
 2. LIMIT(xxx): sets a checking value for USING-derived displacements
 3. WARN(nn): controls USING diagnostics
 - WARN(1): checks for USING “nullification” by other USINGs
 - WARN(2): checks for R0-based USINGs with nonzero base address
 - WARN(4): checks for possible multiple USING resolutions
 - WARN(8): enables checks for resolved displacements exceeding xxx in the LIMIT sub-option
 - WARN values are additive: WARN(15) enables all four checks
- **Check:** recommend USING(WARN(15))

Examples of USING Diagnostics

- Assembler options, including: USING(WARN(15),LIMIT(X'F98'))

```

                                1 START  CSECT
                                00000  2      USING *,10
                                00000  3      USING *,11      A later USING, but...
** ASMA301W Prior active USING on statement number 2 overridden by this USING

                                00000  4      USING *,9      Another later USING
** ASMA300W USING overridden by a prior active USING on statement number 3

                                00000  6      USING B,0
** ASMA302W USING specifies register 0 with a nonzero
           absolute or relocatable base address

                                00FFA  8      USING *+4090,7
** ASMA303W Multiple address resolutions may result from this USING

000000 4120 BFA0      00FA0  10      LA      2,START+4000
** ASMA304W Displacement exceeds LIMIT value specified
                                00004  12 B      EQU      4
```

Overlapping USING-Range Warning: Simple Case

- Typical warning for overlapping USINGs in prolog/entry code:

```
1 Enter    Start 0
2          Using *,15
3          STM   14,12,12(13)      Save registers
4          LR    11,15             Set local base register in R12
5          LR    12,11             Second base
6          AH    12,HW4096         Add 4096 for second base value
7          B     DoSaves           Skip over constant
8 HW4096   DC    H'4096'          Constant

9          Using Enter,11,12       Provide local addressability
** ASMA303W Multiple address resolutions may result from
           this USING and the USING on statement number 2
10         Drop 15                 Drop R15
```

- First impulse: suppress the warning (??)
 - May not be the best idea...
- Easy to fix: move the 'Drop 15' at statement 10 to precede the 'Using Enter,11,12' at statement 9

USING Range Limits

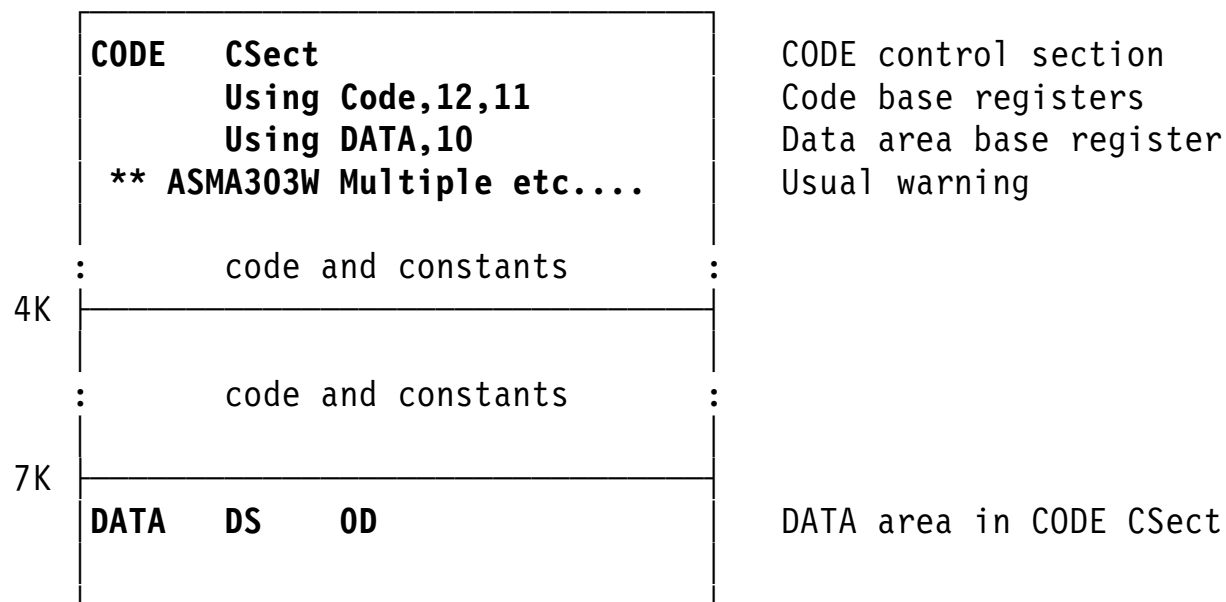
- You may not want a USING range to extend to “full” value
 - Normally, 4096 bytes per base register
- Can limit range by specifying an endloc of allowed range:

USING (baseloc,endloc),regs

- Addressability range restricted to [baseloc,endloc-1]
- endloc may exceed baseloc+4095 without warning
 - Assembler uses the default range [baseloc,baseloc+4095]
 - Except for long-displacement instructions
- Assembler checks for:
 1. baseloc ≤ endloc (ASMA313E if not)
 2. baseloc and endloc having same relocatability attribute (ASMA314E if not)
- Range limits can help eliminate “unavoidable” or accidental range overlaps

Overlapping USING-Range Warning: Unavoidable Cases

- Typical program structure: separate code and data areas



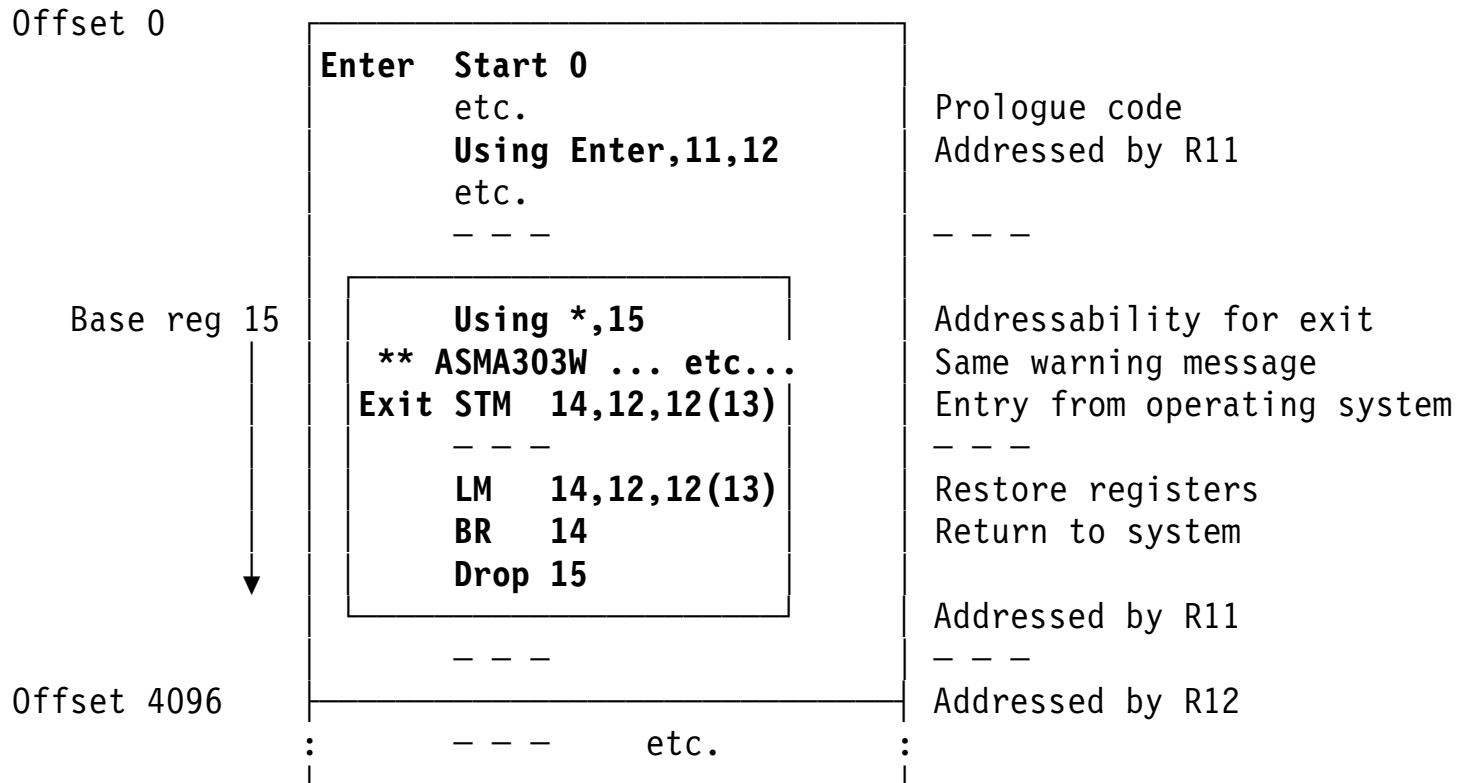
- ASMA303W: USING ranges overlap for code and data base registers
- Solution: specify a *range limit* for the code base

```
USING (CODE,DATA),12,11
```

- Range of first USING does not overlap that of the second!

Overlapping USING-Range Warning: Complex Example

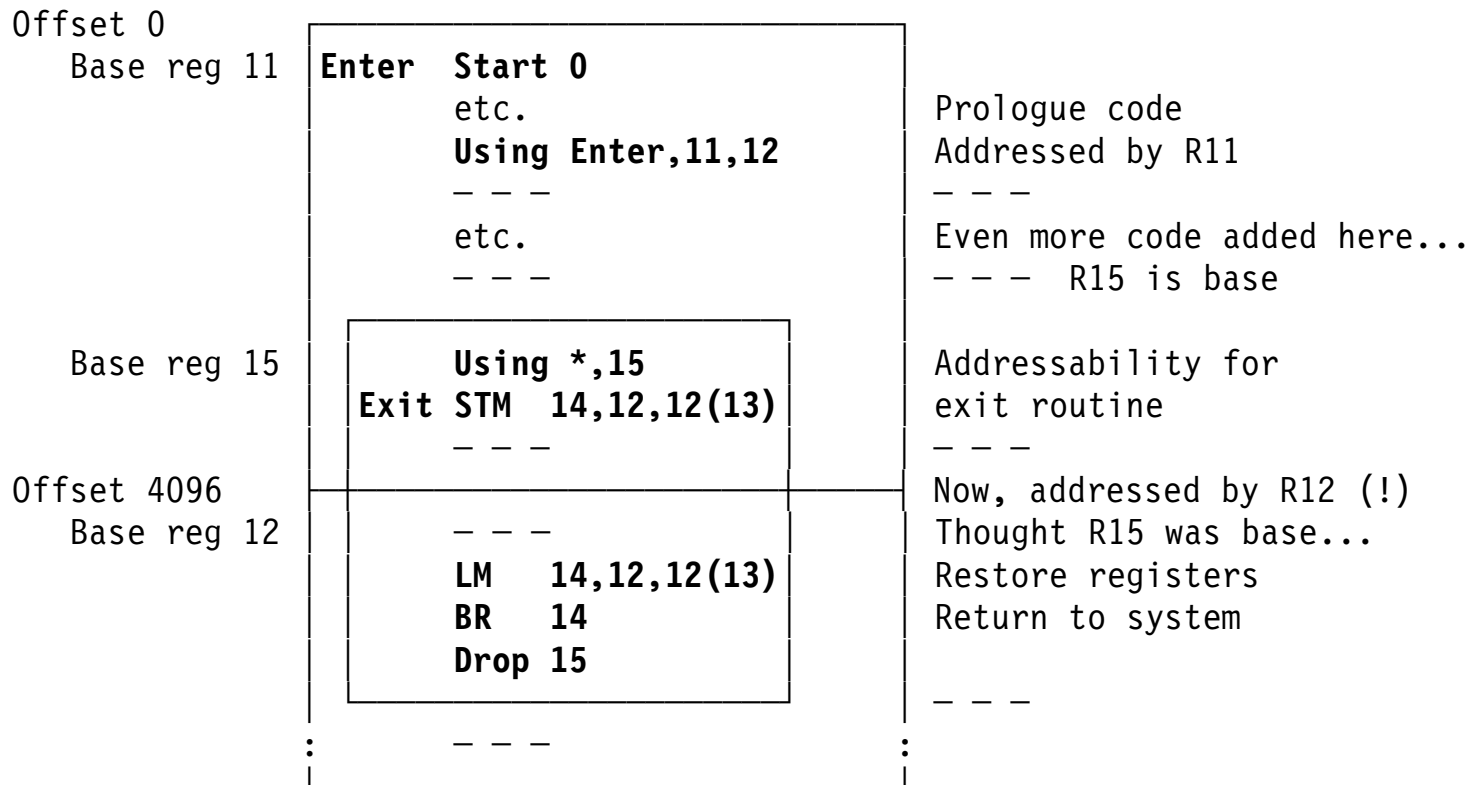
- Program has grown larger, and now has an “asynchronous exit”



- Originally assembled without HLASM: range overlap wasn't flagged!

Overlapping USING-Range Warning: Complex Example, Enhanced

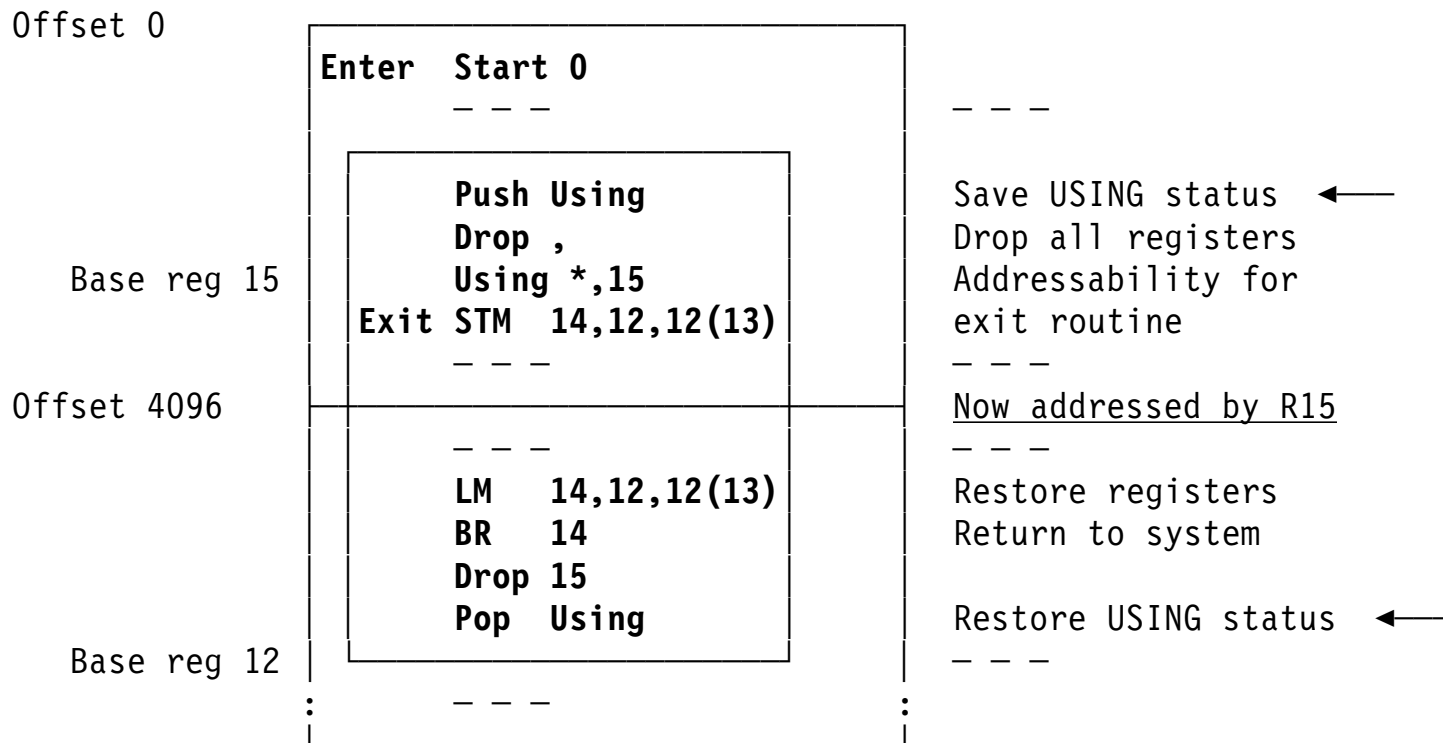
- Program grows; exit starts near offset 4096; warning suppressed



- Wrong base register(s) used in part of the exit!

Fixing Overlapping USING Ranges: Complex Example

- Ensure that only R15 is a base in the exit routine:



- Use PUSH and POP USING to save and restore USING environment

Other Helpful and Informative Diagnostics

- ASMA019W flags length attribute references to symbols having none:

```
000000          B      EQU   *
000000          ...    DS    CL99
          00063    LB     EQU   *-B
          ...
0000DE D200 F063 F000 ...    MVC  A(L'LB),B    L'LB moves only one byte!
** ASMA019W Length of EQUated symbol LB undefined; default=1
```

– You can't use L'B either!

- ASMA031E flags inconsistency between immediate an operand and the instruction:

```
...0000 0000 ... NIHH 0,-16      Operand is inconsistent with operation
** ASMA031E Invalid immediate or mask field
```

```
...0000 0000 ... TML 0,-16      Operand is inconsistent with operation
** ASMA031E Invalid immediate or mask field
```

TYPECHECK Option

- Two suboptions: MAGNITUDE and REGISTER

- MAGNITUDE requests checking of immediate operands

```
... A728 FFF0 ... LHI 2,X'FFF0'      Operand overflows arithmetically
** ASMA320W Immediate field operand may have incorrect sign or magnitude
```

```
ACONTROL TYPECHECK(NOMAGNITUDE)
```

```
... A728 FFF0 ... LHI 2,X'FFF0'      Operand overflows arithmetically
```

- REGISTER checks register/instruction consistency;
you assign Assembler attributes to register symbols

- Helps detect unexpected or inconsistent register references

```
                23          ACONTROL TYPECHECK(REGISTER)
                24 R0      Equ  0,,,GR32
                25 GR0     Equ  0,,,GR64
00002C 4800 F02C   26      LH   GR0,X      (Did you mean LGH?)
** ASMA323W Symbol GR0 has incompatible type with general register field
                27 FRO     Equ  0,,,FPR
000030 5800 F030   28      L    FRO,Y      (Did you mean LE? LD?)
** ASMA323W Symbol FRO has incompatible type with general register field
```

- Generated code may be OK, but results may not be!

- **Check:** inconsistent immediate operands, instruction/register conflicts

Macros and Conditional Assembly

Options and statements to help you find macro-related problems:

- LIBMAC option: puts library macro definitions into the source stream
- Useful PCONTROL sub-options: GEN, MCALL, MSOURCE
 - PRINT-statement operands can also be overridden (slides 19, 20)
- MXREF option provides extensive data on macro/COPY use
- FLAG(SUBSTR) option checks for improper substrings
- COMPAT sub-options: LITTYPE, MACROCASE, SYSLIST (see slide 43)
- MHELP instruction: a built-in macro trace and display facility
- ACTR instruction
 - Limits number of conditional branches within a macro or open code
- Mnemonic collisions: macro names vs. machine instructions
- **Check:** library-macro errors; substring errors; mixed-case macro arguments

COMPAT Option

- COMPAT option enforces some “old rules”:
- COMPAT(MACROCASE): **Unquoted** macro arguments converted *internally* to upper case

AbEnd	1,Dump	Mixed-case argument is accepted, converted
MyMac	'Arg'	Quoted, not converted
MyMac	t'x	Not a quoted string: not converted!

NOCOMPAT(MACROCASE): macro arguments must be typed in the expected (upper) case

AbEnd	1,DUMP	Argument must be in upper case
-------	--------	--------------------------------

- COMPAT(SYSLIST): Inner-macro arguments have no list structure

NOCOMPAT(SYSLIST): Inner-macro arguments may have list structure

COMPAT(SYSLIST) Option

- Old assemblers pass these two types of argument differently:

MYMAC	(A,B,C,D)	Macro call with one (list) argument
&Char SetC	'(A,B,C,D)'	Create argument for MYMAC call
MYMAC	&Char	Macro call with one (string) argument

- Second macro argument was treated simply as a string, not as a list

- Constructed lists may be passed as structures

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

- Can use assembler's full scanning power in all macros
 - No distinction between directly-passed and constructed-string arguments
 - Simplifies logic of inner macros
- COMPAT(SYSLIST) option enforces “old rules”
 - Inner-macro arguments treated as having no list structure

Other Topics

- ACONTROL statement
- Non-invariant characters (@, #, \$, and many others)
- I/O Exits
- SYSADATA files and the ADATA option
 - Full information about all aspects of the assembly
- FOLD option for printed (listing) output
 - Lower case characters are converted (“folded”) to upper case
 - Provides readable output for case-sensitive printers (e.g. Kana)
- Conditional-assembly external functions
- SYSUT1 block size considerations may or may not apply
 - Starting with R5, HLASM defaults to using no work file
 - Specify WORKFILE option for extremely large assemblies
- Attribute references, literals, and Lookahead Mode
- Abnormal terminations

ACONTROL Statement

- **ACONTROL** operands allow changing selected options dynamically
 - COMPAT, LIBMAC, TYPECHECK, FLAG (except RECORD,PUSH), RA2, AFPR
- COMPAT: details on slide 43
- FLAG: details on slide 25

```
L      0,X          X is not on a fullword boundary
** ASMA033I Storage alignment for X unfavorable
ACONTROL FLAG(NOALIGN)
L      0,X          X still not on a fullword boundary; no message
```

- LIBMAC: Lets you accurately locate errors in library macros
- AFPR: controls recognition of Additional Floating Point Registers

```
ACONTROL AFPR      Allow Additional Floating Point Registers
LE      1,=E'6.7'  Float Register 1
ACONTROL NOAFPR    No AFPRs allowed
LE      1,=E'6.7'  Float Register 1
** ASMA029E Incorrect register specification
```

- TYPECHECK: immediate operands, register checks (slide 41)

Attribute References, Literals, and Lookahead Mode

- Symbol attribute reference extensions and enhancements
 - Length, scale, integer attributes allowed in open code
 - Possible errors if old syntax looks like a new attribute reference
- Literals treated more like ordinary symbols
 - May be indexed; offsets allowed
- Lookahead mode: symbol attributes for conditional assembly
 - HLASM “looks ahead” in input file to determine needed attributes
 - Some macro-time attributes may be changed by later symbol definition
 - Cannot “see” any generated statements; scans only source/COPY text

Assembler Abnormal Terminations

Several conditions can cause abnormal or early assembly termination:

- HLASM is unable to load certain modules
 - Main processing module (ASMA93), default options, exits, functions, messages, translate table, Unicode tables
- A loaded module is found to be invalid
- Missing required file(s)
- Invocation-option errors and the PESTOP install option
- External functions and I/O exits
 - ABENDs will kill the assembly
 - Return codes can request explicit (and orderly) termination
- Insufficient virtual storage
 - Specify the WORKFILE option
 - If SYSUT1 is used, some other error situations may be correctable with larger SYSUT1 block size
- Internal or I/O errors (e.g., messages 950-64, 970-1, 976)
- COPY loops: excess DASD or CPU time

COPY Loops and Time/DASD Overruns

- COPY loops can be caused by AIF/AGO instructions in COPY files
- Example: COPY segment named CPYSEG

```
        DC      CL33' '  
        AIF     (&SEGTEST).SKIP  
        DC      C'More stuff'  
.SKIP DC      XL2'0'
```

- If COPY CPYSEG appears more than once in open code...
 - First occurrence of .SKIP defines the sequence symbol
 - Second occurrence of a successful AIF branch goes backward!
- HLASM blindly copies CPYSEG over, and over, and over, and...
- No diagnostic messages:
 - The source listing isn't produced until after conditional assembly is complete
- Remedies:
 1. Put ACTR 20 (or so) at the front of the program
 2. **Always** embed COPY files containing conditional-branch logic inside a macro

Summary

HLASM provides...

- Helpful information:
 - Cross-references for symbols, registers, DSECTs, macros and COPY segments
 - A map of all USING/DROP activity
- Tools for handling possible problems:
 - Diagnostics for programming oversights
 - Options to provide additional checking
 - Options to control the assembler's handling of old code
 - Ways to trace and locate unusual errors
 - Language extensions providing detailed management of USINGS
- Localized controls over assembly-time behavior
 - ACONTROL statement

Let HLASM do what it can to help you!