Finding/Fixing Assembler Language Problems:

How High Level Assembler Can Help

SHARE 102 (February 2004), Session 8173

John R. Ehrman
ehrman@us.ibm.com or ehrman@vnet.ibm.com
IBM Silicon Valley (nee Santa Teresa) Lab
555 Bailey Avenue
San Jose, California 95141 USA

February, 2004

Table of Contents

Table of Contents

Overview: How HLASM Can Help

- Things HLASM <u>can</u> 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 <u>can't</u> help with: (Sorry!)
  - Problems with program structure, logic, or style
    - HLASM Toolkit components can help with these
  - Problems with using the wrong files (such as libraries)
  - Resource constraints (but HLASM can sometimes cope)

Information in the Listing

- Options Summary
- Assembler Service Status (INFO)
- External Symbol Dictionary (ESD)
- Source and Object Code
  - Active-USINGs Heading
- Relocation Dictionary (RLD)
- Ordinary Symbol and Literal XREF
  - Unreferenced Symbols in CSECTs
- Macro and COPY Code Summary
  - Macro and COPY Code XREF
- DSECT XREF
- USING Map
- General Purpose Register XREF
- 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)) ←— ASMAHL command
  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(PRTEXIT(PRTX))
  - - -   etc.
```

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

Assembler Service Status (INFO Option)

- HLASM prints its service status, other useful information
- 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.

19981104 APAR PQ21028 Fixed
         Some machine opcodes incorrectly no longer accept literal operands.

19990113 APAR PQ22004 Fixed
         The message ASMA138W is being issued at the end of a compile when a
         PUSH/POP stack is not empty. The option FLAG(NOPUSH) is provided to
         allow this message to be disabled.
```

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

External Symbol Dictionary (ESD Option)

- The external names defined and referenced by this assembly
  - Normally in upper-case letters
  - 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 are SD, CM, PC)
    - PC sections may cause MODE problems, even if zero length
  - Entry point definitions (type LD)
    - LD-ID points to 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
    - Other products (such as PL/I, binders/loaders) call it “PR”
- 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

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
- Statements and options affecting the source and object code listing
  - PRINT instructions control various portions of the listing
  - PCONTROL can override PRINT-instruction controls (see slide 18)
  - USING and FLAG control various diagnostics (see slides 18, 28)
- To suppress the source and object code listing
  - Selectively: use PRINT instruction operands (see slide 17)
  - Completely: use NOLIST option (but it suppresses the entire listing!)
- **Check:** code in correct sections; END-nominated execution entry

Diagnostic Messages and Severities

- All messages prefixed with `'** ASMA'`
- Final letter of `ASMAnnnS` 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

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 P pointer)
- Flags:

  1. **Type**: whether it's an A-type (data), V-type, Q-type (PR offset), or CXD (PR "Cumulative Length")
  2. **Length**: the constant's length (in bytes)
  3. **Direction**: for A-type adcons, whether to add or subtract the relocation value

- **Check:** intended relocatable items; overlapping RLDs

Ordinary Symbol and Literal Cross-Reference (XREF Option)

- XREF has three sub-options:
  - XREF(FULL) for all symbols, referenced or not
  - XREF(SHORT) for referenced symbols only
  - XREF(UNREFS) for unreferenced non-DSECT symbols
    - Ignored if XREF(FULL) is specified
- Displays information about each symbol:
  - Symbol, length attribute, value
  - Relocation ID, relocatability tags, symbol type, where defined
  - References, including tags indicating use:
    Branch, Drop, Modification, Using, eXecute
- See the example on slide 10
  - 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 literals

Ordinary Symbol and Literal Cross-Reference Example

```
Symbol       Len  Value    Id    R Type Defn  References
  ...
Batch_Init     2 00035C 00000001     H   493  399B  490B      ←— note B tag
Batch_Len      8 000018 FFFFFFFF A   U   772  497
Batch_Msg_1   39 0006A1 00000001     C   681  469   469   716
  ...
Err_Buff       1 000000 FFFFFFFD     J   787  127U  517M  789
Err_Msg      255 000000 FFFFFFFD     C   788  277M  397M  469M  476M  481M  486M  ...
ESD_Amt        2 00000A FFFFFFFC     Y   797  347
ESD_Data      48 000010 FFFFFFFC     C   800  348
ESD_ESD        3 000001 FFFFFFFC     C   795  342   383
ESD_Item       1 000000 FFFFFFFB     J   804  349U  831       ←— note U tag
  ...
Move_Msg       6 0004A6 00000001     I   645  641X            ←— note X tag
  ...
  ...                                            ┌─────┬───── note U,D tags
  ...                                            ↓     ↓
R1             1 000001 00000001 A   U    53  123U  128D  132M  133M  133   135   ...
                                              215D  248M  262   348M  349U  373M  ...
                                              460   560M  574M  578M  582M  586M  ...
  ... etc.
```

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

- MXREF option has three sub-options:
  - MXREF(SOURCE) 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 listing
  - COPY-reference statement numbers tagged with `'C'`
- MXREF data also written to SYSADATA file
  - ASMAXADA sample ADATA exit summarizes “Bill of Materials” info
- **Check:** files from correct libraries; inner macro's callers; duplicate COPY

DSECT Cross-Reference (DXREF Option)

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

```
Dsect     Length      ID       Defn

AEFNPARM 0000001C  FFFFFFFF     165  (negative ID for internal dummy section)
AEFNRIL  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 provides complete summary of all USING/DROP activity:
- Statement-location data
  - Statement number of the USING or DROP
  - Active Location Counter and section ID where the statement appeared
- The type of action requested (USING, DROP)
- Type of USING (Ordinary, Labeled, Dependent, Labeled Dependent)
- Base address, range, and ID of each USING
- Anchoring register on which the USING instruction is based
- Maximum displacement and last statement resolved based on this USING
- The operand-field text of the USING instruction
- **Check:** max displacement; last resolved statement; un-DROPped regs

General Purpose Register Cross-Reference (RXREF Option)

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

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

- **Check:** low utilization; localized loads/stores; based branches

Assembly Summary

- Diagnostic summary: statement, origin, severity
  - Pointers to origins of source statements
- Assembler and host system data
- All files used, I/O and exit counts
- External function statistics
- I/O exit statistics
- Storage utilization data, file-I/O record counts
- Assembly start/stop and processor time info
- **Check:** I/O exits; I/O counts; correct library file ordering; storage use; CPU time

Assembler Options and Diagnostics

- TERM: strongly recommended; always displays a one-line summary
  - Messages displayed (if not suppressed by FLAG option) whether or not PRINT-suppressed in the listing
  - Two suboptions: WIDE (no compression), NARROW (compress blanks)
- BATCH: multiple assemblies with one HLASM invocation
  - Note possible “module contamination”
- PCONTROL: many suboptions (see slide 18)
  - Useful for “exposing” hidden listing information
- FLAG: controls various useful diagnostics (see slide 21)
- USING: controls diagnostics, USING Map (see slide 28)
- LANGUAGE: Select national language for messages, headings
- LIST(133): Wider listing provides more detail
- **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 printing 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!

PCONTROL Option

- PCONTROL lets you override PRINT operands without source changes
  - You can see full details that might have been hidden
- Sub-options are exactly the same as PRINT instruction operands! (Compare slide 17)
  - ON, OFF
  - DATA, NODATA
  - GEN, NOGEN
  - MCALL, NOMCALL
  - MSOURCE, NOMSOURCE
  - UHEAD, NOUHEAD
- GEN, MCALL, MSOURCE useful for macro problems

PCONTROL(MCALL) Option

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

```
    Macro
    TOP    &a,&b,&c
    MIDDLE &c,&a,&b
    MEnd
```

```
         Macro
         MIDDLE &x,&y,&z
&n       SetA   &x*&y+3*&z
         BOTTOM &n
         MEnd
```

```
    Macro
    BOTTOM  &j
    MNote  '&j'
    MEnd
```

- When the TOP macro is invoked with NOMCALL active, no inner calls are visible:

```
 *Process PControl(NoMCALL)
        TOP    2,3,5
+19
```

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

```
 *Process PControl(MCALL)
        TOP    2,3,5
+       MIDDLE 5,2,3
+       BOTTOM 19
+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: FLAG Options

- `FLAG(severity)` controls which messages are printed 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 assembler's `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!
   ** 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 initially
  - Control scope of checking with ACONTROL instructions (see slide 41)

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?
000OA4 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:

```
                          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
```

  - Length attribute of 1+A is 1:

```
0000CC D200 F064 F000 ...      MVC   1+A,B      Moves one byte
** ASMA169I Implicit length of symbol 1+A used for operand 1
```

- Using implicit lengths **is** a good thing!
- **Check:** instruction length fields are assembled correctly

FLAG(PAGE0) Option

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

```
*!     BR   R14        was intended...
       B    R14        Branch to location 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 location 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 location 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
** 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
```

FLAG(PUSH) Option

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

```
Active Usings (1): ...etc...
```

"(1)" indicates USING Push depth = 1

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

FLAG(USING0) Option

- Helps catch accidental use of absolute base address
- Examples of USINGs with absolute base addresses that overlap the assembler's implicit `USING 0,0`

```
                            USING 12,12
  ** ASMA306W USING range overlaps implicit USING 0,0

4110 000A                   LA    1,10
4110 C008                   LA    1,20
```

  – 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

- Message not controlled by an option:

`ASMA308W`	Repeated register in USING

- Messages controlled by the USING(WARN(nn)) option (see slide 28)

`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 specified limit

- Message controlled by the FLAG(USING0) option (see slide 26):

`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:
- MAP: controls Using Map in the listing (see slide 13)
- LIMIT(xxx): sets a checking value for USING-derived displacements
- 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

  WARN values are additive
- **Check:** recommend assembling with USING(WARN(15))

Examples of USING Diagnostics

- Assembler options included `USING(WARN(15),LIMIT(X'F98'))`

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

                         00000     4        USING *,9
  ** 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

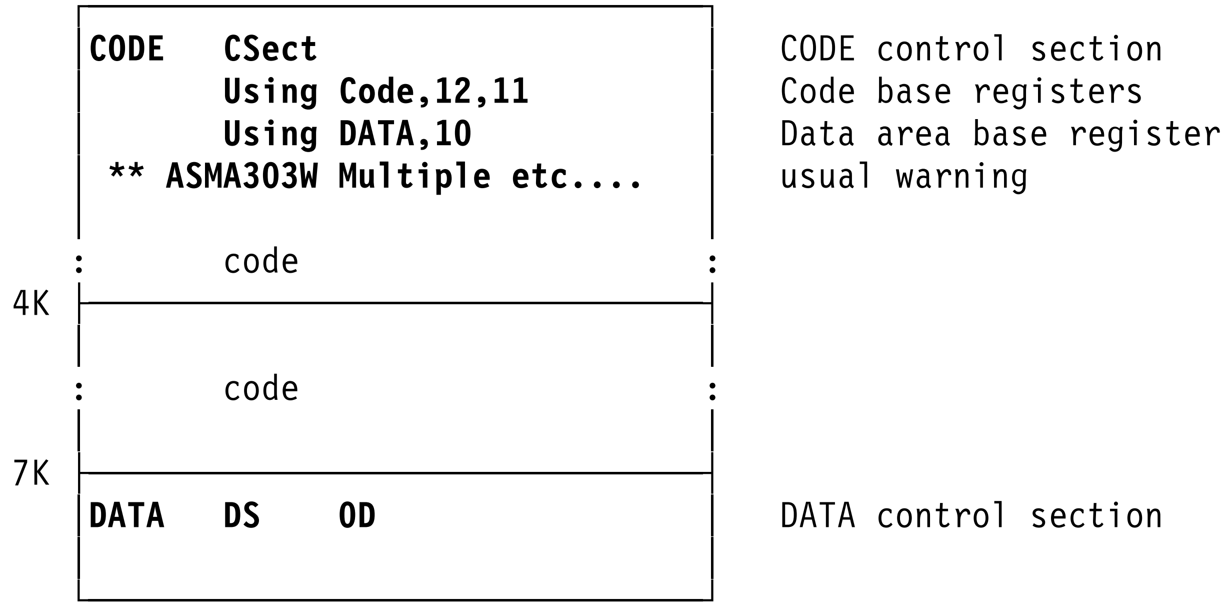
- May not want USING range to extend to “full” value
  - Normally, 4096 bytes per base register
- Can limit range by specifying an endloc of allowed range:

```
          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`]
- Assembler checks for:
  - `baseloc` ≤ `endloc` (ASMA313E if not)
  - `baseloc` and `endloc` have same relocatability attribute (ASMA314E if not)
- Range limits can help eliminate “unavoidable” overlaps

Overlapping USING Range Warning: Unavoidable Cases

- Typical program structure: separate code and data areas



```
    CODE   CSect                           CODE control section
           Using Code,12,11                Code base registers
           Using DATA,10                   Data area base register
     ** ASMA303W Multiple etc....          usual warning

   :       code
4K ├────────────
   :       code
7K ├────────────
    DATA   DS    OD                        DATA control section
```

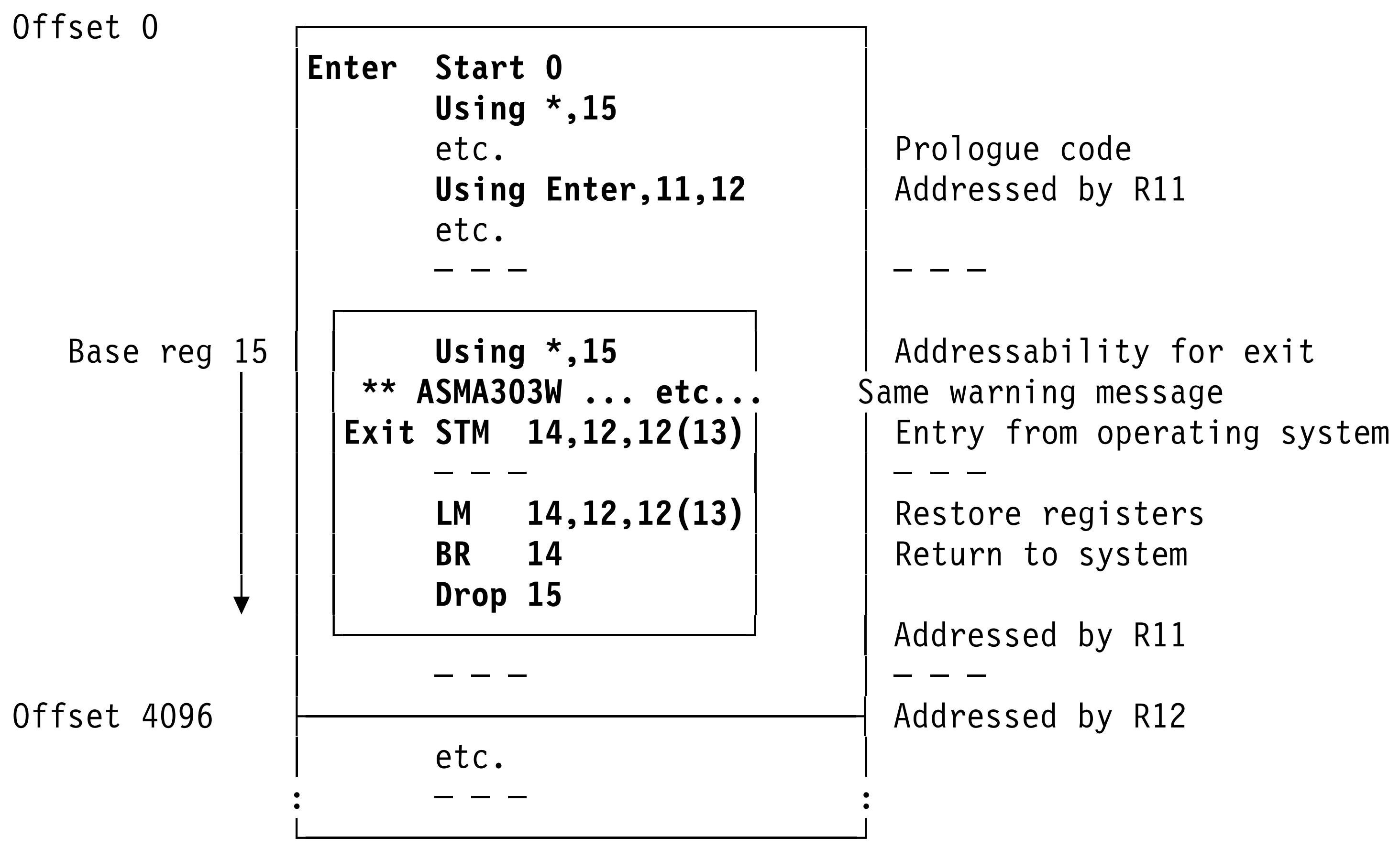
- USING ranges overlap intentionally for code and data base registers
- Solution: specify a *range limit* for the code base

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

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

Overlapping USING Warning: Complex Example

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



```
Offset 0
                  Enter  Start 0
                         Using *,15
                         etc.                  Prologue code
                         Using Enter,11,12     Addressed by R11
                         etc.
                         — — —                 — — —

  Base reg 15            Using *,15            Addressability for exit
                   ** ASMA303W ... etc...     Same warning message
                    Exit STM  14,12,12(13)     Entry from operating system
                         — — —                 — — —
                         LM   14,12,12(13)     Restore registers
                         BR   14               Return to system
                         Drop 15
                                               Addressed by R11
                         — — —                 — — —
Offset 4096                                    Addressed by R12
                         etc.
                         — — —
```

Overlapping USING Warning: Complex Example, Enhanced

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

```
Offset 0
              Enter  Start 0
                     Using *,15
                     etc.                  Prologue code
                     Using Enter,11,12     Addressed by R11
                     - - -                 - - -
                     etc.                  More code added here...
                     - - -                 - - -

   Base reg 15       Using *,15            Addressability for
                Exit STM  14,12,12(13)     exit routine
                     - - -                 - - -
Offset 4096                                Addressed by R12
   Base reg 12       - - -                 Thought R15 was base...
                     LM   14,12,12(13)     Restore registers
                     BR   14               Return to system
                     Drop 15
                                           - - -
             :       - - -              :
```

Fixing Unavoidably Overlapping USINGs

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

```
Offset 0     ┌──────────────────────────────┐
             │Enter  Start 0                │
             │       Using *,15             │
             │       - - -                  │  - - -
             │ ┌───────────────────────┐    │
             │ │     Push Using        │    │  Save USING status
             │ │     Drop ,            │    │  Drop all registers
  Base reg 15│ │     Using *,15        │    │  Addressability for
             │ │Exit STM  14,12,12(13) │    │  exit routine
             │ │     - - -             │    │  - - -
Offset 4096  ├─┼───────────────────────┼────┤  Now addressed by R15
             │ │     - - -             │    │  - - -
             │ │     LM   14,12,12(13) │    │  Restore registers
             │ │     BR   14           │    │  Return to system
             │ │     Drop 15           │    │
             │ │     Pop  Using        │    │  Restore USING status
  Base reg 12│ └───────────────────────┘    │  - - -
             :       - - -                  :
             └──────────────────────────────┘
```

Other Helpful and Informative Diagnostics

- ASMA019W flags length attribute reference to symbols with none:

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

- ASMA031E flags inconsistency between immediate operand and the instruction:

```
... A708 FFF0  ...  LHI   0,-16          Operand is arithmetically valid
... 0000 0000  ...  LHI   1,65520        Operand overflows arithmetically
** ASMA031E Invalid immediate or mask field

... 0000 0000  ...  NIHH  0,-16          Operand is inconsistent with operation
** ASMA031E Invalid immediate or mask field
... A524 FFF0  ...  NIHH  2,X'FFF0'      Operand is logically valid

... A711 FFF0  ...  TML   1,65520        Operand is valid as a mask
```

Macros and Conditional Assembly

- Various options and statements to help find macro-related problems
- LIBMAC option: puts library macro definitions into the source stream
- Useful PCONTROL sub-options: GEN, MCALL, MSOURCE
  - PRINT operands can also be overridden (slides 17, 18)
- MXREF option (see slide 11)
- FLAG(SUBSTR) option (see slide 21)
- COMPAT sub-options: LITTYPE, MACROCASE, SYSLIST (see slide 38)
- MHELP instruction
  - Built-in assembler trace and display facility
- ACTR instruction
  - Limits number of conditional branches within a macro
- **Check:** library-macro errors; substring errors; mixed-case macro arguments

COMPAT Option

- COMPAT option enforces “old rules”:
- `COMPAT(LITTYPE)`: Literal macro operands always have type `'U'`

  `NOCOMPAT(LITTYPE)`: The correct type attribute of the literal constant is used
- `COMPAT(MACROCASE)`: Unquoted macro arguments converted to upper case

  ```
        AbEnd 1,Dump     Mixed–case argument is accepted
  ```

  `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 <u>all</u> 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 instruction
- Non-invariant characters (`@, #, $`)
- I/O Exits
- SYSADATA files and the ADATA option
  - Full information about all aspects of the assembly
- FOLD option for printed (listing) output
  - Lowercase characters are converted (“folded”) to uppercase
  - Provides readable output for case-sensitive printers (e.g. Kana)
- Conditional assembly external functions
- SYSUT1 block size considerations
- Attribute references and Lookahead Mode
- Abnormal terminations

ACONTROL Instruction

- **ACONTROL** operands allow changing selected options dynamically
  - Operands: COMPAT, FLAG (except REC), LIBMAC, RA2, AFPR
- COMPAT: see slide 38 for details
- FLAG: see slide 21 for details

```
       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
- RA2: Tolerate relocatable two-byte address constants
- 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
```

Attribute References, Literals, and Lookahead Mode

- Symbol attribute reference extensions and enhancements
  - Scale, integer attributes allowed in open code
  - Possible errors if old syntax looks like an attribute reference
- Literals treated more like ordinary symbols
  - May be indexed; offsets allowed
- Attribute references to literals are treated more uniformly
  - Previously, could get different results depending on statement ordering (see slide 38)
- Lookahead mode: symbol attributes for conditional assembly
  - HLASM “looks ahead” in input file to determine needed attributes

Assembler Abnormal Terminations

Several conditions can cause abnormal termination of an assembly:

- HLASM is unable to load certain modules
  - Main processing module (ASMA93), default options, opcodes, exits, functions, messages, translate table, Unicode table
- 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
  - Return codes can request explicit (and orderly) termination
  - ABENDs will kill the assembly
- Insufficient virtual storage
- Internal errors (e.g., messages 950-64, 970-1, 976)
  - Some may be correctable with larger SYSUT1 block size
- 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   (&TEST).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 <u>backward</u>!
- HLASM blindly copies CPYSEG over, and over, and over, and...
- No diagnostic messages:
  - The listing isn't produced until <u>after</u> the assembly is done
- Remedies:
  1. Put ACTR 20 (or so) at the front of the program
  2. Embed COPY files with conditional logic inside a macro (always)

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!