High Level Assembler Release 4: New/Updated Features and Functions

SHARE 102 (Feb. 2004), Session 8164

John R. Ehrman
ehrman@vnet.ibm.com or ehrman@us.ibm.com

IBM Silicon Valley (nee Santa Teresa) Laboratory
555 Bailey Avenue
San Jose, CA 95141

February, 2004

New Features in HLASM R4: Overview 1
New Data Types and Constants 2
New and Enhanced Options: CODEPAGE 4
New and Enhanced Options: THREAD 5
Other New and Enhanced Options 6
External Options File 8
Listing Enhancements for Options 9
New and Enhanced Statements 10
New Machine-Instruction Mnemonics 12
New USING-Statement Resolution Rules 15
New Extended Mnemonics 16
XATTR Statement 18
XATTR Statement Syntax 19
XATTR Statement Operand Descriptions 20
R-Type Address Constants and PSECTs 22
New and Enhanced Diagnostics 23
Other Enhancements and Updates 25
Toolkit Feature Enhancements 26
Structured-Programming Macro Enhancements 28
High Level Assembler Release 4 Summary 30
Recommended Service 31
Related Sessions at SHARE 32
References 33

- New data types and constants
- New options
- New instructions
- New statements
- Listing enhancements
- New and enhanced diagnostics and messages
- Other assembler enhancements
- Toolkit Feature enhancements
- HLASM documentation also available from the HLASM web site:

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

- Address constants

AD 8-byte doubleword-aligned address constant

  - Note: may not necessarily be supported in other products

Expressions evaluated to 32 bits, sign-extended to 64

```
000010 FFFFFFFFFFFFFFA1                6       DC    AD(-95)
000018                                 7       DC    AD(-123456789012)
** ASMA146E Self-defining term too long or value too large -- 123456789012)
```

Non-relocatable part must be in range $(-2^{31}, 2^{31}-1)$

R **PSECT** address (more about this later)

- Binary constants

  FD 8-byte doubleword-aligned binary integer

  Same as currently-supported FL8, but aligned

```
000000 000000000000005F          4     DC    FD'95'
000008 FFFFFFE34166E5EC          5     DC    FD'-123 456 789 012'
```

  Nominal value in the range $(-2^{63}, +2^{63}-1)$

- Character constants

  CU 16-bit Unicode-encoded character data

  – Single-byte characters converted automatically to Unicode™

  – Nominal-value character representation specified by **CODEPAGE** option

- **CODEPAGE**(codepage_identifier)
  - Specifies (in decimal or hex) the code page used for encoding single-byte data in CU-type constant nominal values
  - Supported code pages (all include the “euro”):

Dec	Hex	Country Character Sets
1140	X'0474'	US, CA, NL, AU, NZ, Portugal, Brazil
1141	X'0475'	Austria, Germany
1142	X'0476'	Denmark, Norway
1143	X'0477'	Finland, Sweden
1144	X'0478'	Italy
1145	X'0479'	Spain, Latin America (Spanish)
1146	X'047A'	United Kingdom
1147	X'047B'	France
1148	X'047C'	International–1 (Default)

  - Usage example with code page 1148:

```
000000 0053004800410052              2      DC    CU'SHARE 95'
000008 0045002000390035
```

  - Mapping-table structure documented in Programmer's Guide

- Location counter values have always been “threaded” – each control section starts at the next doubleword boundary following the previous
  - Example with **THREAD** option:

```
000000                  00000 00008     1 Sect1  Start 0
000000 000000000000005F                 2        DC    FD'95'

000008                  00008 00008     3 Sect2  CSect
000008 FFFFFFFFFFFFFFA9                 4        DC    AD(*-95)
                                        5        End
```

    - A tradition from the old days of absolute loaders (like CMS's)
      - Assembled locations matched loaded addresses (like CMS's X'20000')
  - **NOTHREAD** option resets each subsequent section origin to zero (simplifies calculation of offsets)

```
000000                  00000 00008     1 Sect1  Start 0
000000 000000000000005F                 2        DC    FD'95'

000000                  00000 00008     3 Sect2  CSect
000000 FFFFFFFFFFFFFFA1                 4        DC    AD(*-95)
                                        5        End
```

- OPTABLEs **ZOP, YOP** described shortly
- OPTABLE(xxx,**LIST**) lists opcode table xxx
  - Implemented by APAR PQ44437
- Example:

High Level Assembler Operation Code Table Contents

Mnemonic	Frmt	HexOP	Operands	Mnemonic	Frmt	HexOP	Operands	Mnemonic	Frmt	HexOP	Operands
ACONTROL	HLASM			ACTR	HLASM			AD	RX	6A	R1,D2(X2,B2)
ADATA	HLASM			ADB	RXE	ED1A	R1,D2(X2,B2)	ADBR	RRE	B31A	R1,R2
ADR	RR	2A	R1,R2	AE	RX	7A	R1,D2(X2,B2)	AEB	RXE	ED0A	R1,D2(X2,B2)
– – – – – etc.											
UPT	E	0102		USING	HLASM			WXTRN	HLASM		
X	RX	57	R1,D2(X2,B2)	XATTR	HLASM			XC	SS	D7	D1(L,B1),D2(B2)
XG	RXE	E382	R1,D2(X2,B2)	XGR	RRE	B982	R1,R2	XI	SI	97	D1(B1),I2
XR	RR	17	R1,R2	ZAP	SS	F8	D1(L1,B1),D2(L2,B2)				

- **GOFF**: synonym for existing **XOBJECT** option
  - Not the same as C/C++'s XOBJ!
- **FLAG([NO]EXLITW)** controls warning message ASMA016W
  - Implemented by APAR PQ67377

```
000000 4400 F008          000008      4   EX  0,=X'07FE'
 ** ASMA016W Literal used as the target of EX instruction
```

- **[NO]TYPECHECK**: error ASMA031E changed to warning ASMA320W
  - Before applying APAR PQ69375:

```
000000 0000 0000          000000      3   LHI 0,X'FFFF'
 ** ASMA031E Invalid immediate or mask field
```

  - After applying APAR PQ69375:

```
                                      3   ACONTROL NOTYPECHECK
000000 A708 FFFF           0FFFF      4   LHI   0,X'FFFF'
                                      5   ACONTROL TYPECHECK
000004 A708 FFFF           0FFFF      6   LHI   0,X'FFFF'
 ** ASMA320W Invalid immediate operand
```

- External options file
  - DDname ASMAOPT contains options (MVS, CMS)
    - File may be fixed or variable length records
  - Eliminates limitations on JCL PARM strings
- Options precedence hierarchy (highest to lowest):
  0. Fixed installation defaults (specified with DELETE operand of the OPTIONS installation macro)
  1. ***PROCESS OVERRIDE** options
  2. Options in the ASMAOPT file (or VSE Librarian member ASMAOPT.USEROPT)
  3. Options in JCL PARM (MVS, VSE) or ASMAHL command (CMS)
  4. Options on JCL OPTION statement (VSE)
  5. Options on ***PROCESS** statements
  6. Non-fixed installation defaults
- Listing shows origins of option overrides
  - Conflicting overrides produce severity-2 “Notification”

- Listing shows options-hierarchy origins for overrides

```
Overriding ASMAOPT Parameters – NODXREF,NODECK        ◄— ASMAOPT file
Overriding Parameters–  asa,noobj                     ◄— 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
NOEXIT
– – –   etc.
```

- Numeric tags in left margin indicate the origin of the override

- ***PROCESS** statement **OVERRIDE** operand
  - Lets you make certain options for a source file “unchangeable”

```
*PROCESS OVERRIDE(CODEPAGE(X'047B'))  All CU-type constants are in French!
```

  - Avoids problems of accidental or inaccurate PARM option specification
- Additional extended branch mnemonics
  - For relative branch instructions (see slide 16)
- New z/Architecture machine instructions in OPTABLEs UNI, ZOP, YOP
  - **OPTABLE(ESA)** option may help avoid macro conflicts (e.g. **MSG**)
    (Or, use macros with names longer than 5 characters)
- ***Note!*** **OPTABLE(ESA)** means “ESA machines,” not the z/Architecture instructions that execute in ESA mode

- **AMODE** and **RMODE** operand extensions

  AMODE ANY31, 64

  RMODE 31, 64

  Some operands not necessarily meaningful to other products!

- **XATTR** assigns attributes to external symbols (more at slides 18-22)
- **ALIAS** operand extended to 256 bytes (APAR PQ57792)
  - Previous limit was 64 bytes for external symbols (**GOFF** only)
    - Internal symbols still limited to 63 characters

- Machine instructions (in **OPTABLE(ZOP)**; APAR PQ44437)

```
  AG       AGF      AGFR     AGHI     AGR      ALC      ALCG     ALCGR    ALCR     ALG
  ALGF     ALGFR    ALGR     BCTG     BCTGR    BRASL    BRCL     BRCTG    BRXHG    BRXLG
  BXHG     BXLEG    CDGBR    CDGR     CDSG     CEGBR    CEGR     CG       CGDBR    CGDR
  CGEBR    CGER     CGF      CGFR     CGHI     CGR      CGXBR    CGXR    *CLCLU    CLG
  CLGF     CLGFR    CLGR     CLMH     CSG      CSP      CVBG     CVDG     CXGBR    CXGR
  DL       DLG      DLGR     DLR      DSG      DSGF     DSGFR    DSGR     EPSW     EREGG
  ESEA     ICMH     IIHH     IIHL     IILH     IILL     LARL     LCGFR    LCGR     LCTLG
  LG       LGF      LGFR     LGHI     LGR      LLGC     LLGF     LLGFR    LGH      LLGH
  LLGT     LLGTR    LLIHH    LLIHL    LLILH    LLILL    LMD      LMG      LMH      LNGFR
  LNGR     LPGFR    LPGR     LPQ      LPSWE    LRAG     LRV      LRVG     LRVGR    LRVH
  LRVR     LTGFR    LTGR     LURAG    MGHI     ML       MLG      MLGR     MLR      MSG
  MSGF     MSGFR    MSGR    *MVCLU    NG       NGR      NIHH     NIHL     NILH     NILL
  OG       OGR      OIHH     OIHL     OILH     OILL    *PKA     *PKU      RLL      RLLG
  SAM24    SAM31    SAM64    SG       SGF      SGFR     SGR      SLAG     SLB      SLBG
  SLBGR    SLBR     SLG      SLGF     SLGFR    SLGR     SLLG     SRAG     SRLG     STCMH
  STCTG    STFL     STG      STMG     STMH     STPQ     STRAG    STRV     STRVG    STRVH
  STURG    TAM      TMHH     TMHL    *TP       TRACG   *TROO    *TROT    *TRTO    *TRTT
 *UNPKA   *UNPKU    XG       XGR
```

- Extended Translation Facility-2 instructions (*)

- z990 Machine instructions (in **OPTABLE(YOP)**; APAR PQ74561)
- Message-Security Assist (ESA/390 also!)

  KM KMC KIMD KLMD KMAC

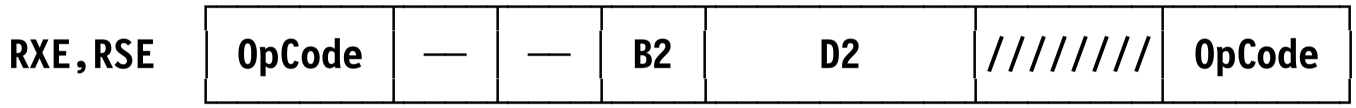
- HFP Multiply-and-Add/Subtract Facility(ESA/390 also!)

  MAD MADR MAE MAER MSD MSDR MSE MSER

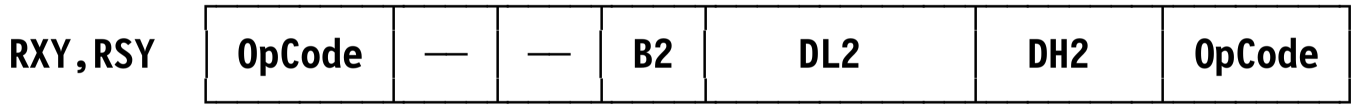
- DAT Enhancement Facility

  CSPG IDTE

- Long-displacement instructions:



- The 5th byte (////////) of z900 instructions used for signed displacement extension DH2 in z990



- Long-Displacement Facility: 44 ***new*** instructions

```
AHY     ALY     AY      CDSY    CHY     CLIY    CLMY    CLY     CSY     CVBY
CVDY    CY      ICMY    ICY     LAMY    LAY     LB      LDY     LEY     LGB
LHY     LMY     LRAY    LY      MSY     MVIY    NIY     NY      OIY     OY
SHY     SLY     STAMY   STCMY   STCY    STDY    STEY    STHY    STMY    STY
SY      TMY     XIY     XY
```

  – All instructions end with "**Y**," hence opcode table named YOP

- Long-Displacement Facility: 69 ***enhanced*** instructions

```
AG      AGF     ALC     ALCG    ALG     ALGF    BCTG    BXHG    BXLEG   CDSG
CG      CGF     CLCLU   CLG     CLGF    CLMH    CSG     CVBG    CVDG    DL
DLG     DSG     DSGF    ICMH    LCTLG   LG      LGF     LGH     LLGC    LLGF
LLGH    LLGT    LMG     LMH     LPQ     LRAG    LRV     LRVG    LRVH    ML
MLG     MSG     MSGF    MVCLU   NG      OG      RLL     RLLG    SG      SGF
SLAG    SLB     SLBG    SLG     SLGF    SLLG    SRAG    SRLG    STCMH   STCTG
STG     STMG    STMH    STPQ    STRV    STRVG   STRVH   TRACG   XG
```

- z990: 128 new and enhanced instructions

- Long-displacement instructions support 20-bit **signed** displacement
- HLASM resolution rules for base-displacement resolution:
  (changed rules are underlined)
  1. Expression and USING-table entry relocatability attributes must match
  2. Calculate possible displacements; choose smallest non-negative value
  3. If no positive displacements are available, use smallest negative value
     (z/900-z/990 differences require non-negative displacement if possible)
  4. If more than one smallest displacement, choose higher-numbered register

```
000000                  00000 00012    1 Test CSect ,
                  R:AB  00000          2      Using *,10,11
                        00000          3 X    Equ   *
000000 E300 B880 1208         13880    4      AG    0,X+80000  Long displacement
000006 E300 AFA0 0008         00FA0    5      AG    0,X+4000   R11 +96 bytes away
                                       6      Drop  10
00000C E300 BFA0 FF08         00FA0    7      AG    0,X+4000   Negative displacement
                                       8 *    Note absolute displacements:
000000 E300 0120 7A71          7A120   9      LAY   0,+500000
000006 E300 0000 8371         F83000  10      LAY   0,-512000
```

- Extended and alternate mnemonics for “Long” relative branches
  - Analogs of extended mnemonics for “short” relative branches
  - All based on BRCL; implemented by APAR PQ44437
  - **COPY IEABRC** for macros to replace based branches with relative

BROL		**JLO**		**Mask=1**	**“Branch Relative Long if Ones”**
BRHL	**BRPL**	**JLH**	**JLP**	**Mask=2**	**“Jump Long if High”**
BRLL	**BRML**	**JLL**	**JLM**	**Mask=4**	
BRNEL	**BRNZL**	**JLNE**	**JLNZ**	**Mask=7**	
BREL	**BRZL**	**JLE**	**JLZ**	**Mask=8**	
BRNLL	**BRNML**	**JLNL**	**JLNM**	**Mask=11**	
BRNHL	**BRNPL**	**JLNH**	**JLNP**	**Mask=13**	
BRNOL		**JLNO**		**Mask=14**	
BRUL		**JLU**		**Mask=15**	**“Jump Long Unconditionally”**

- More extended and alternate branch mnemonics

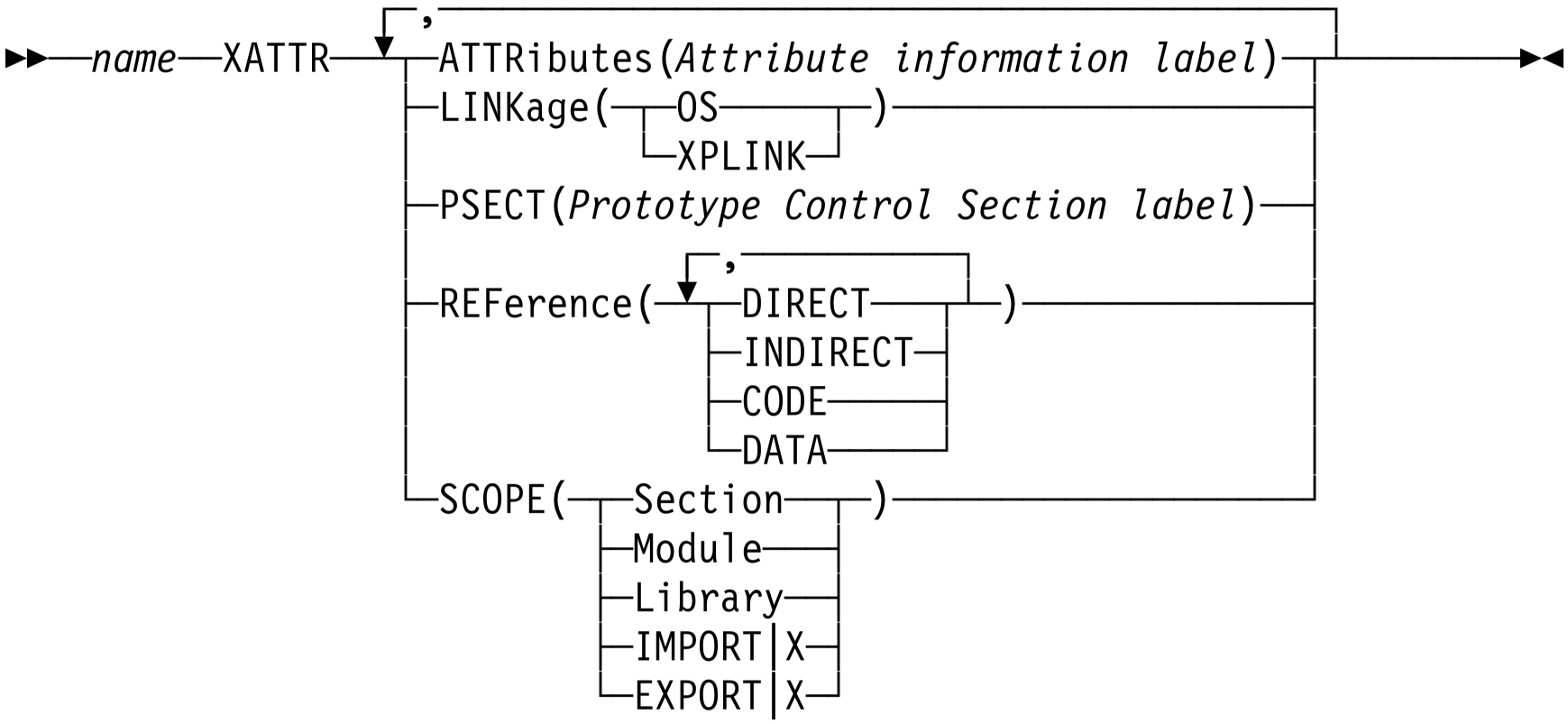
Extended	Base Instruction	
JAS	BRAS	Branch Relative and Save
JASL	BRASL	Branch Relative and Save Long
JCT	BRCT	Branch Relative on Count (32–bit)
JCTG	BRCTG	Branch Relative on Count (64–bit)
JXH	BRXH	Branch Relative on Index High (32–bit)
JXHG	BRXHG	Branch Relative on Index High (64–bit)
JXLE	BRXLE	Branch Relative on Index Low or Equal (32–bit)
JXLEG	BRXLG	Branch Relative on Index Low or Equal (64–bit)
JLNOP	BRCL 0	Long Relative No–Op

  - Implemented by APAR PQ51476

- Assigns eXternal-symbol ATTRibutes (requires **GOFF** option)
  - For definitions and references
  - Declarations used at bind time for resolutions, diagnostics
  - Currently used only for XPLink
- Supported attributes are:

ATTRIBUTES	Specifies location of extended attribute data
LINKAGE	Type of linkage: OS or XPLINK
PSECT	Locates an element in a non-shared class; a “Prototype Section”
REFERENCE	Direct or Indirect; Code or Data
SCOPE	Binding scope: SECTION, MODULE, LIBRARY, EXPORT/IMPORT

- All attribute declarations for a symbol must be on a single XATTR statement



- **ATTRIBUTES**:
  - Provides “extended” attribute data about the symbol

```
MyFunc   XATTR  Attributes(Attr_data)   'Attr_data' is a label in module's TXT
```

  - Extended attributes supply additional info not needed at bind time
    - Extended-attribute data may reside in a different section and class from that of the external symbol itself
  - A binder-defined, architected data structure
- **LINKAGE**:
  - The symbol calls, or is called, using XPLink conventions

```
ExtFunc  XATTR  Linkage(XPLink)  If omitted, default assumed to be OS linkage
```

- **PSECT**:
  - Connects shared code in which the symbol is defined to a non-shared work area associated with the code

```
MyFunc   XATTR  PSect(Work_area_label)
```

  - **MyFunc**: an entry or section name in shared code
  - **Work_area_label**: a label in an element in a non-shared Read/Write class

- **REFERENCE**:
  - References from or to the symbol may be direct or indirect (e.g. via a linkage descriptor for XPLink)
  - The symbol defines or references code or data

```
ExtFunc  XATTR  Reference(Indirect,Code)
```

  - The binder does not currently check for inconsistent code/data declarations
- **SCOPE**:
  - The binder will resolve the symbol within its Section, the bound Module (no library search), using Library search, or is eXported/Imported for DLL use

```
          EXTRN  DLLFunc          Declare "DLLFunc" external
DLLFunc   XATTR  Scope(Import)    Declare it having IMPORT status
```

  - Note: "Scope-X" symbols are "Import" if declared **EXTRN**, "Export" if declared **ENTRY, CSECT**, or **RSECT**
  - Generalization of existing scope support:
    - Load modules support module scope (**WXTRN**) and library scope (**EXTRN**)

- Shared code typically requires a non-shared work area
  - The non-shared area is called a Prototype Section (PSect)
  - Not a “control section” in the traditional sense
- Need to establish
  1. the association of the shared and non-shared sections
     - The **PSECT** operand of **XATTR** connects the two:

```
Shared      CSECT  ,                        Shared-code Section
Code_Class  CATTR  EXECUTABLE,RENT          Shared-code Class
            - - -                       ...Read-only code and constants
Shared      XATTR  PSect(Non_Shared)        Declare PSect for shared code
PS_Class    CATTR  NOTREUS,NOTEXECUTABLE    PSect Class (working storage)
            - - -                       ...Initialized read/write storage
Non_Shared  DC     9D'0'                    Label in PSect Class
```

  2. a pointer to a copy of the non-shared area for each invocation of the shared code
     - Provided by the R-type address constant:

```
PSectPtr    DC     R(Shared)          Resides in shared-code caller's PSect
```

     - The binder and loader resolve the R-con to the new instance of the non-shared PSect that was associated (via the **XATTR** declaration) with the **Shared** section
- Currently used only for XPLink-related linkages

- Invalid lengths for **MP** and **DP** instructions now detected
  - Example with 5-byte first and second operands:

```
000000 0000 0000 0000 00000 00000     2         DP   Result,=P'12345.6789'
** ASMA069S Length of second operand must be less than first

000006 0000 0000 0000 00000 00000     3         MP   Result,=P'12345.6789'
** ASMA069S Length of second operand must be less than first

00000C                                4 Result DS   PL5
```

  - Explicit zero lengths accepted (assumed to be targets of **EX**)
- Many messages now include information about the erroneous operand

```
000000 0000 0000            00000     2          LA   777,29
** ASMA029E Incorrect register specification — 777
```

  - Most “Delimiter error” messages clarified; also provide operand data

- New "**N**otification" (severity 2) messages for attempts to override parameters already specified at a higher level of the hierarchy

```
Overriding Parameters-  ASA,ESD
Process Statements-     NOESD

** ASMA436N Attempt to override invocation parameter in a *PROCESS statement.
           Option NOESD ignored.
```

- New messages for problems loading the Unicode mapping tables

```
Process Statements-     OVERRIDE(CODEPAGE(X'5555'))

** ASMA945U Unable to load Code Page ASMA5555
```

- Symbolic **DMin** operand for denormalized minimum representable floating point value

```
DC  EH'(DMin)'      generates  X'00000001'    Hex Float
DC  EB'-(DMin)'     generates  X'80000001'    Binary Float
```

  - Completes the set of floating point symbolic values
    - R3 provided **MIN** (normalized), **MAX, NAN, INF**

- Predefined absolute symbols no longer allowed in conditional assembly character expressions
  - Was a potential source of many conditional-assembly ambiguities

```
A       Equ   C'F'          Value of A is 'F'
&C      SetCF A,'Arg'       Formerly, it would call function 'F', not 'A'
```

    - Now the first operand of **SETCF** must be quoted

```
&C      SetCF 'A','Arg'     Call function 'A'
```

  - Many curiosities and surprises were also possible with null bytes...
  - If you don't understand the problems, be glad you didn't have them!

- Toolkit Overview in session **8166**, demo in session **8171**!
- 64-bit debugging support in Interactive Debug Facility (ASMIDF)
  - APAR PQ51325; PTFs: MVS UQ57987, VM UQ57988, VSE UQ57989
  - **REGS64, GPRG, GPRH** commands for displaying/updating 64-bit GPRs
  - **EPNAMES** command displays entry point names window
- Updated (and improved) publications are
  - Interactive Debug Facility User's Guide, GC26-8709-04
  - Interactive Debug Facility Reference Summary, GC26-8712-03
  - Available only in softcopy (at HLASM web site, also)
- **ASMLANGX** performance enhancement (APAR PQ61239)
- Enhanced SuperC:
  - **FINDALL** process option (APAR PQ51367)
  - 31-bit enablement (APAR PQ66218)

- **ASMXREF**: new **SYMC** sort for SWU; 31-bit enabled (APAR PQ67403)
- The disassembler module **ASMADOP** has been updated to:
  1. Disassemble z/Architecture instructions correctly.
     - The coincident vector-facility machine opcodes are no longer displayed:

```
VACDR VADQ  VADR  VAEQ  VAQ   VAR   VCDQ  VCDR  VCEQ  VCQ   VCR
VDDQ  VDDR  VDEQ  VLCDR VLCER VLCR  VLDQ  VLDR  VLEQ  VLMDQ VLMDR
VLMEQ VLMQ  VLNDR VLNER VLNR  VLPDR VLPER VLPR  VLQ   VLZDR VMADQ
VMAEQ VMCDR VMDQ  VMDR  VMEQ  VMQ   VMR   VMSDQ VMSEQ VNQ   VNR
VOQ   VOR   VSDQ  VSDR  VSEQ  VSQ   VSQDR VSQER VSR   VXQ   VXR
```

  2. Correctly display the value of the immediate operand of RIL-format and RIE-format instructions.
- Program Understanding Tool (**ASMPUT**)
  - 30-day free trial version
  - Includes ASMPUT graph-printing facility
  - Interactive demonstration of ASMPUT
- Basic and advanced demos of Interactive Debug Facility (**ASMIDF**)

- Major updates and enhancements
- Support for relative branch instructions: **ASMMREL** macro
  - Parameters ON or OFF select based vs. relative branch instructions
- Macro extensions:
  - **DO** macros accept labels (usable by **ITERATE, ASMLEAVE**)
  - **SELECT** with no operands allows following **WHEN**s with **IF**-type operands
- New macros:
  - **ELSEIF** eliminates unneeded nesting of **IF/ELSE/ENDIF**
  - **ITERATE** – Iterate the current or other enclosing DO loop
  - **ASMLEAVE** – Leave the current or other enclosing DO loop
    - — See next slide...

- Other changes:
  - **CASENTRY** macro uses GPR0 as a temporary register
  - **DOEXIT** can be used in other structures inside a DO group
  - **DO** macro uses **LHI** in preference to **LH** and a literal
  - More checking for proper syntactic nesting
    - For example: ending a **DO** with **ENDIF**
  - **ELSE, OTHRWISE** check for single use in a group
  - Generated labels use **DC OH**, not **EQU ***
  - Global variables use **ASMA_** prefix to avoid conflicts
- Implemented in APAR PQ69812
- **Note:** Updated by APAR PQ74641 to fix IMS problem
  - **LEAVE** macro renamed to **ASMLEAVE**
  - A newly-supplied **ASMNAMES** file can be edited to rename any/all macros
  - PTFs: CMS UQ77271, VSE UQ77281, MVS UQ77253
- Many of these enhancements suggested and/or prototyped by customers

- Support for new hardware and software features
  - New instructions, data types, data conversions
  - Support for 64-bit debugging
- New and enhanced options
  - **CODEPAGE, THREAD**, OPTABLE(**ZOP|YOP|...,LIST**), **TYPECHECK, FLAG(EXLITW)**
- New statements
  - ***PROCESS OVERRIDE, XATTR**; new AD, FD, CU, and R-type constants
- Expanded and enhanced diagnostic support
  - Checks for conflicts among options sources
  - Improved information for error messages
- Toolkit Feature enhancements
  - 64-bit debugging, new Structured-Programming macros
- Many enhancements provided with service!

IBM's continuing commitment to support Assembler applications

- Latest APARs, as of January 2004:
- **HLASM**: PQ82515
- **SuperC**: PQ82387
- **Interactive Debug Facility**: PQ76914
- **ASMDASM**: PQ66807
- **Structured Programming Macros**: PQ74641
- **ASMXREF**: PQ67403
- **ASMPUT**: PQ58466

- Other sessions of possible interest:

Session	Session Title
8116	Introduction to I/O Programming in Assember Language
8162	Program Checks, ABENDs, Dumps: What They Are and How to Stop Suffering From Them
8167-8	Assembler as a High Level Language: Conditional Assembly and Macro Concepts
8117	Stalking the Esoteric OpCodes
8163	HLASM Programming in a Relative Instruction Set World
8166	HLASM Toolkit Feature Overview
8171	HLASM Toolkit Feature Demonstrations
8156	Programming for Data Spaces in HLASM
8162	Stalking the New OpCodes
8132	Extending the Life Cycle of Legacy Applications
8165	HLASM Features: Benefits and Exploitation
8173	How HLASM Helps Find/Fix Assembler Language Problems
8154	Brushing Up on the Assembler Classics

- High Level Assembler for MVS & VM & VSE publications:
  - Language Reference (SC26-4940)
  - Programmer's Guide (SC26-4941)
  - Toolkit Feature User's Guide (GC26-8710)
  - Toolkit Feature Interactive Debug Facility User's Guide (GC26-8709)
  - ...and others of not-as-general interest
- z/OS publications:
  - z/OS MVS Programming: Assembler Services Guide
  - z/OS MVS Programming: Assembler Services Reference