High Level Assembler Release 4: New Features and Functions

SHARE 96 (Feb. 2001), 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 26, 2001

- 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

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

- Binary constants

  FD 8-byte doubleword-aligned binary integer

  Same as currently-supported FL8, but aligned

```
000000 000000000000005F                2        DC    FD'95'
```

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

- 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

```
000008 FFFFFFFFFFFFFFA1                3        DC    AD(-95)
```

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

  R PSECT address (more about this at slide 13)

- Character constants

  CU 16-bit Unicode-encoded character data (see slide 3)

  – Single-byte characters converted automatically to Unicode

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

- **GOFF**: synonym for XOBJECT
  - Not the same as C/C++'s XOBJ!
- **CODEPAGE**(codepage_identifier)
  - Specifies (in decimal or hex) the code page used for encoding single-byte data in CU-type constant nominal values
  - Usage example:

```
000000 0053004800410052                2        DC    CU'SHARE 96'
000008 0045002000390036
```

  - Supported code pages (all include the “euro”):

Dec	Hex	Description
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)

  - Mapping-table structure documented in Programmer's Guide

- **THREAD** and **NOTHREAD**
  - Location counter values are “threaded:” each control section starts at the next doubleword boundary following the previous section.
  - 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 were the same as loaded addresses (like CMS's `X'20000'`)

  - **NOTHREAD** option resets each subsequent section origin to zero

```
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
```

    — Simplifies calculation of offsets

- OPTABLE(xxx,**LIST**) lists opcode table xxx (APAR PQ44437)

- 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 (see slide 7)
  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 (see slide 6)
  - Conflicting overrides produce low-level “notifications” (see slide 14)

- 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
  - See options precedence hierarchy on slide 5
  - Blank field means precedence 0 or 6

- Additional extended branch mnemonics
  - For relative branch instructions (see slide 8)
- New z/Architecture machine instructions in OPTABLEs UNI, ZOP
  - Described in Session 8172 (summary on slide 8)
  - **OPTABLE(ESA)** option may help avoid macro conflicts
    (Or, use macros with names longer than 5 characters)
- `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 9-13)
- ***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 inaccurate PARM option specification

- Implemented by APAR PQ44437
- Extended branch mnemonics for “Long” relative branches

```
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"
```

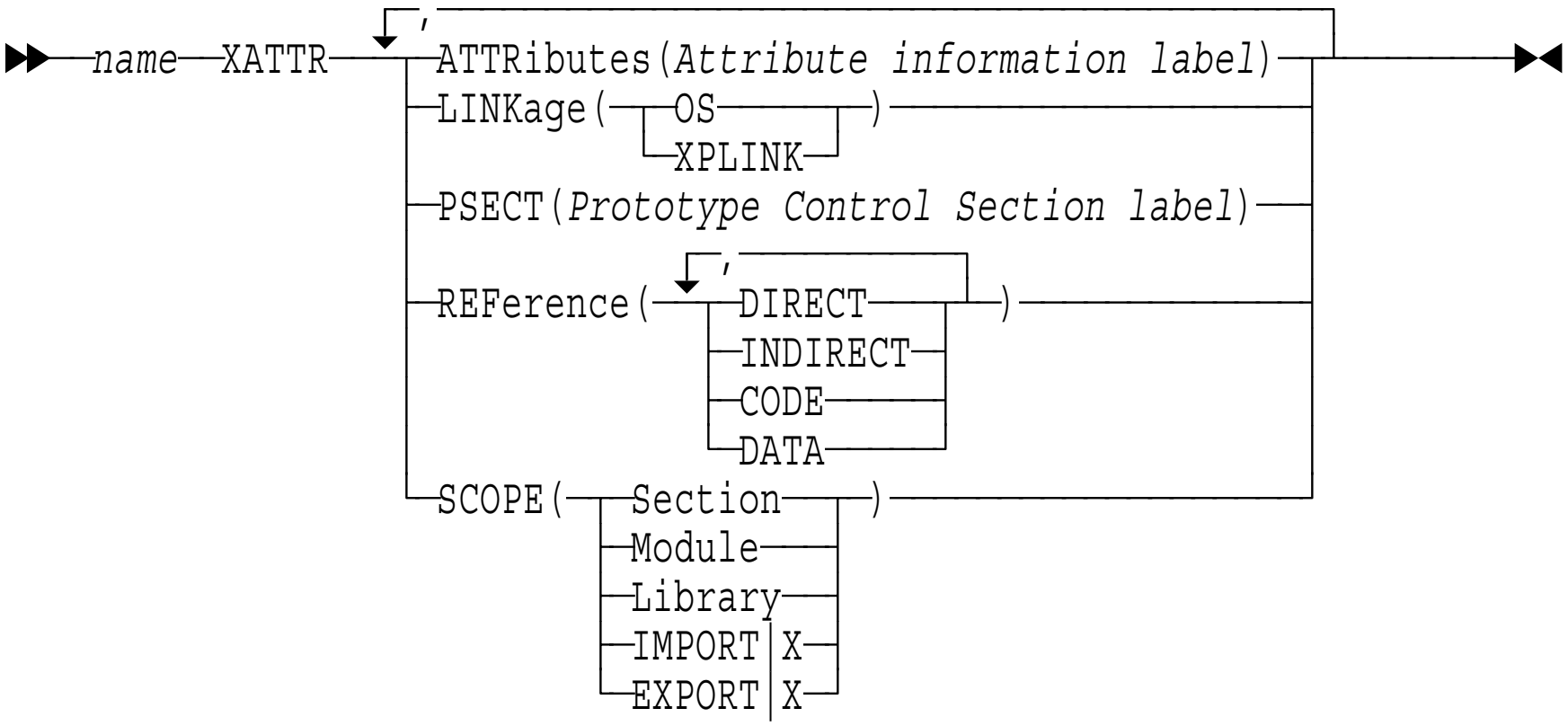
- Machine instructions (* means “not executable just yet”)

```
  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
```

- 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” (see slide 13)
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 is OS linkage
```

- PSECT:
  - Connects shared code in which the symbol is defined to a non-shared work area associated with the code (see slide 13)

```
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 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

```
DLLFunc  XATTR  Scope(Import)
```

  - Generalization of existing scope support:
    - Load modules support module scope (WXTRN) and library scope (EXTRN)

- Each invocation of shared code requires a non-shared work area
  - The non-shared area is called a Prototype Section (PSect)
  - Note: 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
```

- Literals entered into symbol table when encountered
  - Helps avoid problems of needing attribute reference before first use
- Symbolic `DMin` operand for minimum representable floating point value

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

- 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 require the first operand of SETCF to be quoted

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

- Enhanced SuperC supports ALLMEMS option
  - Scans all members, including aliases
- ASMPUT graph-printing facility coming soon (via PTF)
- Items available from the HLASM web site:
  - 30-day free trial version of Program Understanding Tool (ASMPUT)
  - Demonstration of ASMPUT
  - Basic and advanced demonstrations of Interactive Debug Facility (ASMIDF)
- HLASM documentation also available from the web site
  - IDF manual improved

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

- Support for new hardware and software features
  - New instructions, data types, data conversions
- New and enhanced options
  - **CODEPAGE, THREAD**, OPTABLE(...,**LIST**)
- 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

IBM's continuing commitment to support Assembler applications

- 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
- Related sessions at SHARE 96:

Session	Description
8165	HLASM Features: Benefits and Exploitation
8170	New Programming Support in the DFSMS/MVS Binder
8172	New zSeries Instructions: Assembler Programmer's View
8166	HLASM Toolkit Feature Overview
8171	HLASM Toolkit Feature Demonstration