The latest IBM Z COBOL compiler: Enterprise COBOL V6.2!

Tom Ross

Captain COBOL

SHARE Providence

August 7,2017

COBOL V6.2 ? YES!

- The 4th release of the new generation of IBM Z COBOL compilers
- Announced: July 17, 2017
  - The same day as IBM z14 hardware…coincidence?
- GA: September 8, 2017
  - Compiler support for the new IBM z14 hardware and IBM z/OS V2.3
  - Ability to exploit the new Vector Packed Decimal Facility of z14

- 'OLD' news:
  - COBOL V5 EOM Sept 11, 2017 (announced Dec 6, 2016)
  - EOS for COBOL V4 'might' be earlier than 2020, still discussing

COBOL V6.2 ? What else does it have?

- New and changed COBOL statements, such as the new JSON PARSE statement
- Support of COBOL 2002/2014 standards with the addition of the COBOL Conditional Compilation language feature
- New and changed COBOL options for increased flexibility
- Improved compiler listings with compiler messages at the end of the listing as in previous releases of the compiler
- Improved interfaces to optional products and tools such as IBM Debug for z Systems (formerly Debug Tool for z/OS) and IBM Application Discovery and Delivery Intelligence (formerly EzSource)
- Compile-time and runtime performance enhancements
- Improved usability of the compiler in the z/OS UNIX System Services environment

Vector Packed Decimal Facility of z14

- Enterprise COBOL V6.2 adds support for exploiting the new Vector Packed Decimal Facility in z14 through the ARCH(12) compiler option.
- The Vector Packed Decimal Facility allows the dominant COBOL data types, packed and zoned decimal, to be handled in wide 16-byte vector registers instead of in memory.
  - Decimal and floating-point computationally intensive COBOL programs, which are optimized with Enterprise COBOL V6.2 and that target z14 ARCH(12), can deliver CPU time reduction on the z14 server over the same applications built with COBOL V6.1.
  - No source changes are required to take advantage of this new facility; just recompile with ARCH(12) to target z14.

Changed ARCH compiler option

- ARCH(7) (still the default in 6.2)
  - 2094-xxx models (IBM System z9 EC) 2096-xxx models (IBM System z9® BC)
- ARCH(8)
  - 2097-xxx models (IBM System z10 EC) 2098-xxx models (IBM System z10 BC)
- ARCH(9)
  - 2817-xxx models (IBM zEnterprise z196 EC) 2818-xxx models (IBM zEnterprise z114 BC)
- ARCH(10)
  - 2827-xxx models (IBM zEnterprise EC12) 2828-xxx models (IBM zEnterprise BC12)
- ARCH(11)
  - 2964-xxx models (IBM z13)
- **ARCH(12)**
  - **3906-xxx models (IBM z14)**

Example 1 – Unsigned Packed Decimal Add – 4.85x Faster

```
01  WS-VAR-1   COMP-3   PIC s9(7)
 01  WS-VAR-2   COMP-3   PIC s9(7).
 01  WS-VAR-3   COMP-3   PIC s9(7).
 . . .
ADD WS-VAR-1 TO WS-VAR-2
             GIVING WS-VAR-3.
```

Timing (100 million times in a loop)

```
COBOL V4:   3.648 cpu seconds
ARCH(11):   2.195 cpu seconds
ARCH(12):   0.752 cpu seconds
```

ARCH(12) is 4.85 times faster than COBOL V4
ARCH(12) is 2.91 times faster than ARCH(11)
80% less CPU compared to V4!!!!

V4
ARCH(6|7|8|9|10)

- Use in memory instructions
- Explicit sign setting

```
MVC     168(4,R9),160(R9)
OI      171(,R9),X'0F'
MVC     352(4,R13),152(R9)
OI      355(,R13),X'0F'
AP      168(4,R9),352(4,R13)
OI      171(,R9),X'0F'
```

ARCH(11)

- Convert to DFP
- Conversion overhead

```
CDPT    FP0,160(4,R9),0x9
CDPT    FP1,152(4,R9),0x9
ADTR    FP0,FP0,FP1
LPDFR   FP0,FP0
CPDT    FP0,168(4,R9),0xb
```

ARCH(12)

- Use new ARCH(12) facility
- No conversions, no explicit sign setting

```
VLRL    VRF16,160(,R9),0x3
VLRL    VRF17,152(,R9),0x3
VAP     VRF16,VRF16,VRF17,0x7,14
```

Example 2 – Large Decimal Divide – 135x Faster

```
01 WS-VAR-1  COMP-3  PIC s9(29)
01 WS-VAR-2  COMP-3  PIC s9(3).
01 WS-VAR-3  COMP-3  PIC s9(25)v9(6).
. . .
DIVIDE WS-VAR-1 BY WS-VAR-2
    GIVING WS-VAR-3.
```

Timing (100 million times in a loop)

```
COBOL V4 or
COBOL V5/V6 w/ARCH(11):   2.319 cpu seconds
ARCH(12):                 0.027 cpu seconds
```

ARCH(12) is 135 times faster than COBOL V4 (or COBOL V5/V6 with ARCH(11) or less)!
99% less CPU compared to pre-ARCH(12)!!!

Without ARCH(12)

- Call out to LE library routine
- Pre shifting operation
- Piecewise divide, call overhead

```
ZAP   336(16,13),16(2,2)
MVC   352(32,13),58(10)
MVC   366(15,13),0(2)
NI    380(13),X'F0'
MVN   383(1,13),14(2)
L     3,92(0,9
L     15,180(0,3)     V(IGZCXDI )
LA    1,180(0,10
BASR  14,15
```

With ARCH(12)

- Use new ARCH(12) facility
- Inline hardware accelerated shift+divide

```
VLRL    VRF24,_WSA[0x12c] 0(,R3),0xe
VLRL    VRF25,_WSA[0x12c] 16(,R3),0x1
VSDP    VRF24,VRF24,VRF25,0x6,0
```

Example 3 – Large Decimal Multiply – 39x Faster

```
01 WS-VAR-1  COMP-3  PIC s9(14)v9(4).
01 WS-VAR-2  COMP-3  PIC s9(14)v9(4).
01 WS-VAR-3  COMP-3  PIC s9(14)v9(2).
MULTIPLY WS-VAR-1 BY WS-VAR-2
     GIVING WS-VAR-3.
```

Timing (100 million times in a loop)

```
COBOL V4 or
COBOL V5/V6 w/ARCH(11):   2.797 cpu seconds
ARCH(12):                 0.072 cpu seconds
```

ARCH(12) is 39 times faster than COBOL V4 (or COBOL V5/V6 with ARCH(11) or less)!
97.5% less CPU compared to pre-ARCH(12)!!!

Without ARCH(12)

- Call out to LE library routine
- Piecewise multiply, call overhead
- Post shifting operation

```
L      3,92(0,9)
L      15,188(0,3)   V(IGZCXMU )
LA     1,171(0,10)
BASR   14,15
NI     388(13),X'0F'
MVN    396(1,13),399(13)
ZAP    32(9,2),388(9,13)
```

With ARCH(12)

- Use new ARCH(12) facility
- Inline hardware accelerated multiply+shift

```
VLRL     VRF16,152(,R9),0x9
VLRL     VRF17,168(,R9),0x9
VMSP     VRF16,VRF16,VRF17,0x6,0
```

Example 4 – Zoned Decimal Computation – 3.05x Faster

```
01 WS-VAR-1 PIC 9(8) value 1352435.
01 WS-VAR-2 PIC s9(8)v9(2).
01 WS-VAR-3 PIC s9(10)v9(2).
01 WS-VAR-4 PIC s9(8)v9(2).
. . .
COMPUTE WS-VAR-4 = (WS-VAR-1 / 365) *
          (WS-VAR-2 + 1) - WS-VAR-3.
```

Timing (100 million times in a loop)

```
COBOL V4:   1.469 cpu seconds
ARCH(11):   0.837 cpu seconds
ARCH(12):   0.482 cpu seconds
```

ARCH(12) is 3.05 times faster than COBOL V4
ARCH(12) is 1.74 times faster than ARCH(11)
67% less CPU compared to V4!!!!

V4
ARCH(6|7|8|9)

- Use in memory instructions

```
PACK   296(8,13),0(8,2)
SRP    298(6,13),2(0),0
...
DP     296(8,13),40(2,10)
ZAP    264(16,13),296(6,13)
PACK   280(16,13),8(10,2)
...
PACK   296(8,13),24(12,2)
SRP    296(8,13),2(0),0
SP     268(12,13),296(8,13)
...
```

ARCH(10|11)

- Convert to DFP

```
CDZT    FP1,_WSA[0x12c] 0(8,R3),0x8
SLDT    FP0,FP1,2
...
DDTR    FP0,FP0,FP1
FIDTR   FP1,9,FP0
LXDTR   FP0:FP2,0,FP1
CDZT    FP1,_WSA[0x12c] 8(10,R3),0x8
...
MXTR    FP4:FP6,FP0:FP2,FP8:FP10
CXZT    FP0:FP2,_WSA[0x12c] 24(12,R3),0x8
SLXT    FP8:FP10,FP0:FP2,2
SXTR    FP0:FP2,FP4:FP6,FP8:FP10
...
```

ARCH(12)

- Use new ARCH(12) facility

```
VPKZ    VRF24,_WSA[0x12c] 0(,R3),0x7
VSRP    VRF24,VRF24,0xa,0x2,2
...
VLIP    VRF25,0x365,0
VDP     VRF24,VRF24,VRF25,0xa,0
...
VMP     VRF24,VRF24,VRF25,0x15,0
VPKZ    VRF25,_WSA[0x12c] 24(,R3),0xb
VSRP    VRF25,VRF25,0xe,0x2,0
VSP     VRF24,VRF24,VRF25,0x16,0
...
```

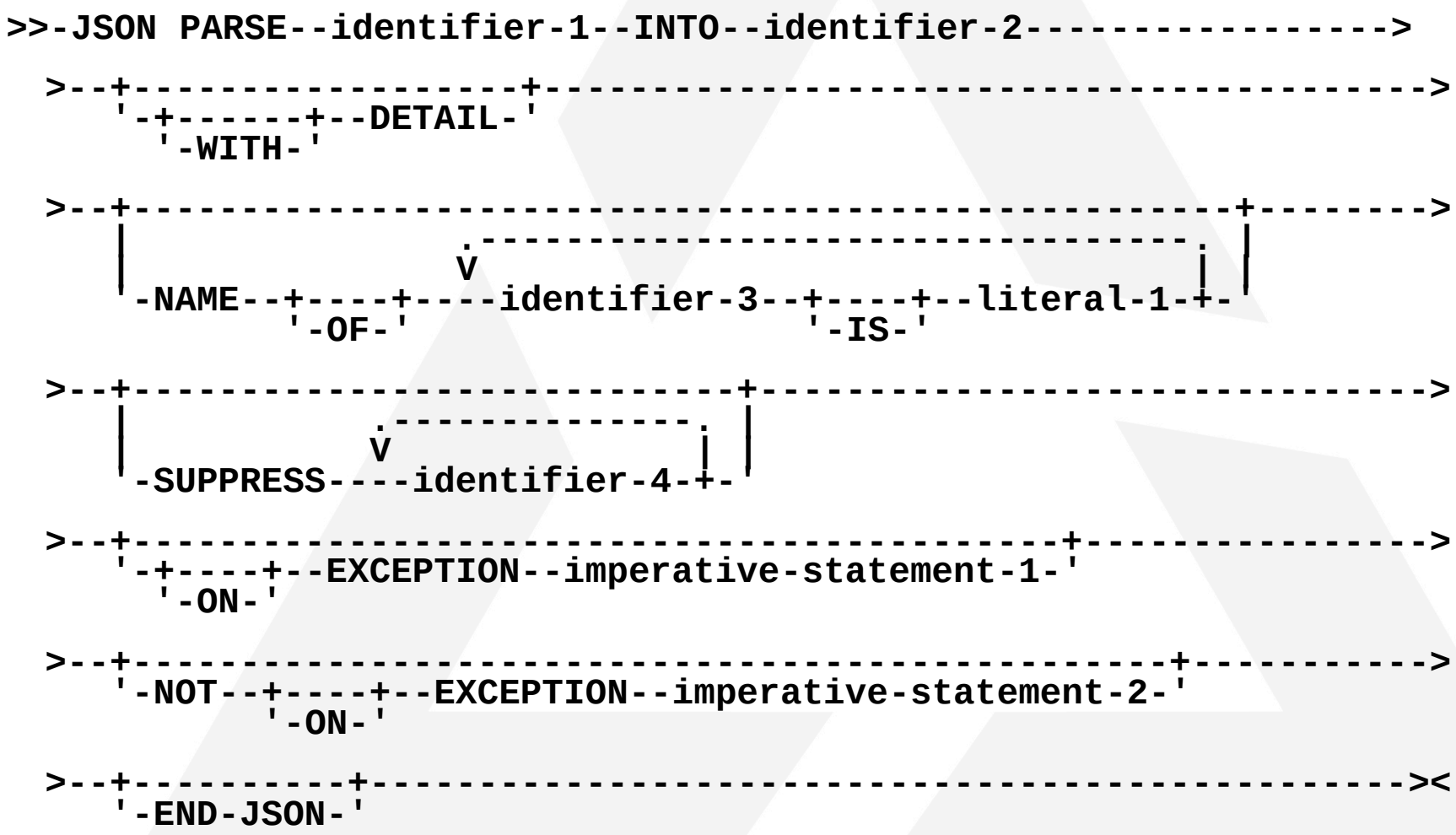
New COBOL Language

- New Statement
  - JSON PARSE … INTO

- New Compiler Directive (to control optimization inlining)
  - >>INLINE
  - >>NOINLINE
- New Compiler Directives (for conditional compilation)
  - >>DEFINE
  - >>EVALUATE
  - >>IF

JSON PARSE … INTO

- The JSON PARSE statement converts JSON text to COBOL data formats.
- Syntax:



JSON PARSE

- JSON PARSE *identifier-1* INTO *identifier-2*
  [[WITH] DETAIL]
  [NAME [OF] {*identifier-3* [IS] *literal-1*}...]
  [SUPPRESS {*identifier-4*}...]
  [[ON] EXCEPTION *imperative-statement-1*]
  [NOT [ON] EXCEPTION *imperative-statement-2*]
  [END-JSON]
- Not so similar to XML PARSE
  - No PROCESSING PROCEDURE
  - Not event-driven
  - INTO *identifier 2* !!

JSON PARSE

- Two kinds of condition might occur during execution of a JSON PARSE statement and might result in the receiver being partially modified:
  - Nonexception conditions result in reason codes in special register JSON-STATUS, but do not terminate execution of the statement.
  - Exception conditions cause execution to be terminated with the exception code set in special register JSON-CODE.
    - Any nonexception conditions that were detected prior to the exception condition are represented in the reason codes set in special register JSON-STATUS.

JSON PARSE

Examples of matched and mismatched data definitions and JSON text

- The following JSON text and data definitions are considered an exact match:

```
JSON PARSE input-json INTO G

{"g": {"H": {"A": "Eh?", "3_": 55, "c-C": "SeeThisLongString"}}}

01 G.
   02 h.
         05 a pic x(10).
       04 3_ pic 9.
     03 C-c pic x(10).
```

Same as

```
01 G.
   02 h.
        03 a pic x(10).
        03 3_ pic 9.
        03 C-c pic x(10).
```

NOTE: Truncation and padding are normal, just like in MOVE

JSON PARSE

- When WITH DETAIL is specified, fixed-point number or string truncation from the previous example would be diagnosed with runtime messages as follows:

IGZ0328I During execution of the JSON PARSE statement on line *line-number* of program *program-name*, assignment of the value of the JSON name/value pair at offset *offset* to data item 3_ resulted in loss of significance ("SIZE ERROR").

IGZ0329I During execution of the JSON PARSE statement on line *line-number* of program *program-name*, assignment of the value of the JSON name/value pair at offset *offset* to data item C-C *data-name* resulted in a loss of information.

JSON PARSE

SHARE
EDUCATE • NETWORK • INFLUENCE

"Missing" names in JSON are tolerated, but matches require the level ("qualification") names to match. For example, the following data definitions and JSON text are compatible, but data-items "3_" and "etc" would not be modified by a JSON PARSE statement (JSON-STATUS would be set to 1):

```
{"g": {"H": {"A": "Eh?", "c-C": "See", "etc:{"etc": "xxx"}}}
```

```
01 G.
   02 h.
         05 a pic x(10).
       04 3_ pic 9.
     03 C-c pic x(10).
   02 etc.
     03 etc pic x(10).
```

JSON PARSE

Superfluous JSON name/value pairs are tolerated, but result in setting special register JSON-STATUS to a condition code, and possibly in one or more runtime messages, under control of the WITH DETAIL phrase.
For example, the following data definitions and JSON text are compatible, and would result in completely populating group G, but JSON text "B": "Bee" would not be used, and the JSON PARSE statement would terminate with a non-exception condition: JSON-STATUS = 2

```
{"G": {"h": {"a": "Eh?", "B": "Bee", "3_": 5, "C-c": "See"}}}
```

```
01 G.
   02 h.
      05 a pic x(10).
      05 3_ pic 9.
      05 C-c pic x(10).
```

Note: The program would not terminate, and all data items populated, so that the ‘flexible’ nature of parameter passing is supported

JSON PARSE

- If superfluous JSON name/value pairs are found, and WITH DETAIL was specified, then the program will get the following runtime message (in addition to getting JSON-STATUS set to 2:

  IGZ0321I During execution of the JSON PARSE statement on line *line-number* of program *program-name*, no data item matched JSON name *JSON-name* at offset *offset*.

JSON PARSE

The following data definitions and JSON text are fully compatible, despite the name order mismatch.

```
{"g": {"H": {"3_": 5, "A": "Eh?", "c-C": "See"}}}
```

```
  01 G.
       05 h.
          10 a pic x(10).
          10 3_ pic 9.
          10 C-c pic x(10).
```

JSON PARSE

The following data definitions and JSON text are not a match, because of the omitted name level (qualification by “H”), and would result in an exception condition, JSON-CODE=106, with identifier-2 (“G”) unchanged:

```
JSON PARSE input-json INTO G WITH DETAIL

 {"g": {"A": "Eh?", "3_": 5, "c-C": "See"}}

01 G.
   02 h.
         05 a pic x(10).
       04 3_ pic 9.
     03 C-c pic x(10).

Use WITH DETAIL to get runtime messages with
explanation of what, if any, conditions occurred
```

JSON PARSE

- If WITH DETAIL was specified, and the previous incompatible JSON text and COBOL data definitions are used, then the program will get the following message at run time, in addition to getting JSON-CODE set to 106:

  IGZ0341W During execution of the JSON PARSE statement on line *line-number* of program *program-name*, no JSON name/value pair matched any data item in the receiver. The receiver G was not modified.

JSON PARSE

The following data definitions and JSON text are not compatible, because the JSON values are all incompatible with the corresponding data items, and would result in an exception condition, with identifier-2 (“G”) unchanged:

```
{"g": {"H": {"A": 42, "3_": "x", "c-C": "123"}}}
```

```
01 G.
   02 h.
         05 a pic a(10).
       04 3_ pic 9.
     03 C-c pic 99.
```

New and changed COBOL statements

- Support of COBOL 2002/2014 standards with the addition of the COBOL Conditional Compilation language feature
  - New compiler directives
    - **>>DEFINE**
    - **>>EVALUATE, >>WHEN, >>END-EVALUATE**
    - **>>IF, >>ELSE, >>END-IF**
  - New compiler option
    - **DEFINE**
- Conditional compilation provides a way of including or omitting selected lines of source code depending on the values of literals specified by the DEFINE directive. In this way, you can create multiple variants of the same program without the need to maintain separate source streams.

Conditional Compilation

Set compiler option **DEFINE(iscics=B'01')**

```
DEFINE iscics AS PARAMETER

>>IF iscics
        EXEC CICS HANDLE ABEND
>>ELSE
        CALL 'CEEHDLR' USING myhandler, etc
>>END-IF
```

Conditional Compilation

Set compiler option **DEFINE(subsys='IMS')**

```
DEFINE subsys AS PARAMETER

>>EVALUATE TRUE
  >>WHEN subsys='BATCH'
        CALL 'DB2BATCH'
  >>WHEN subsys='CICS'
        CALL 'DB2CICS'
  >>WHEN subsys='IMS'
        CALL 'DB2IMS'
>>END-EVALUATE
```

Conditional Compilation

- Predefined compilation variables
  - Variables that are defined automatically by the compiler.
  - These compilation variables, listed below, can be referenced in conditional compilation directives wherever a compilation variable is allowed.

Predefined compilation variable name	Value
ARCH	Numeric value representing the value of the ARCH compiler option in effect.
CICS	B'1' if CICS compiler option is in effect, B'0' otherwise.
COMPILER-VRM	Numeric value representing the compiler version, release, and mod level (e.g., 610 is COBOL V6.1.0)
DLL	B'1' if DLL compiler option is in effect; B'0' otherwise.
DYNAM	B'1' if DYNAM compiler option is in effect; B'0' otherwise.
OPTIMIZE	Numeric value representing the value of the OPTIMIZE compiler option.in effect
SQL	B'1' if SQL compiler option in effect, B'0' otherwise.
SQLIMS	B'1' if SQLIMS compiler option is effect, B'0' otherwise.
THREAD	B'1' if THREAD compiler option is effect, B'0' otherwise.

Conditional Compilation

```
*>  Use predefined compilation variables
*>  These only reflect compiler options, could have more
*>  more than one, like both SQL and CICS!
>>EVALUATE TRUE                          *> Check which coprocessor
  >>WHEN SQL                             *> DB2
        EXEC SQL blah blah END-SQL
  >>WHEN CICS
        EXEC CICS blah blah END-CICS
  >>WHEN SQLIMS
        EXEC SQLIMS blah blah END-SQLIMS
>>END-EVALUATE
```

Examples of Conditional Compilation

SHARE
EDUCATE • NETWORK • INFLUENCE

Example 4: use OVERRIDE and OFF in the DEFINE directive

```
>>DEFINE VAR AS 12          *> Sort of like 77 VAR pic 99 value 12.
. . .
>>DEFINE VAR OFF            *> VAR is now gone/undefined!
. . .
>>IF VAR IS DEFINED
     compute x = x + 1      *> This code will not be included
>>ELSE
     compute x = x - 1      *> This code will be included
>>END-IF
. . .
>>DEFINE VAR AS 16          *> VAR is back!
. . .
>>DEFINE VAR AS VAR - 2 OVERRIDE   *> VAR is now 14
. . .
>>IF VAR IS EQUAL TO 16
     compute x = x + 1      *> This code will NOT be included
>>ELSE
     compute x = x - 1      *> This code will be included
>>END-IF
```

New and changed COBOL options

- The IBM-supplied default for the AFP compiler option is changed from VOLATILE to NOVOLATILE
  - So that the compiler can generate more efficient code sequences for programs with floating point operations.
- New DEFINE option allows you to define or set conditional compilation constants at compile time.
- New INITCHECK option tells the compiler to perform a static analysis of the program, and to emit a warning message for data items that are used before they are initialized.
- Features to control inlining behaviors at OPTIMIZE(1) or OPTIMIZE(2):
  - Sometimes inlining is not desired, like for error paragraphs that are rarely executed
  - INLINE compiler option tells compiler to allow inlining of procedures if OPT decides it makes sense, NOINLINE tells compiler to not allow it
  - >>INLINE ON and >>INLINE OFF compiler directives to disable specific procedures within the program from being inlined.

New and changed COBOL options

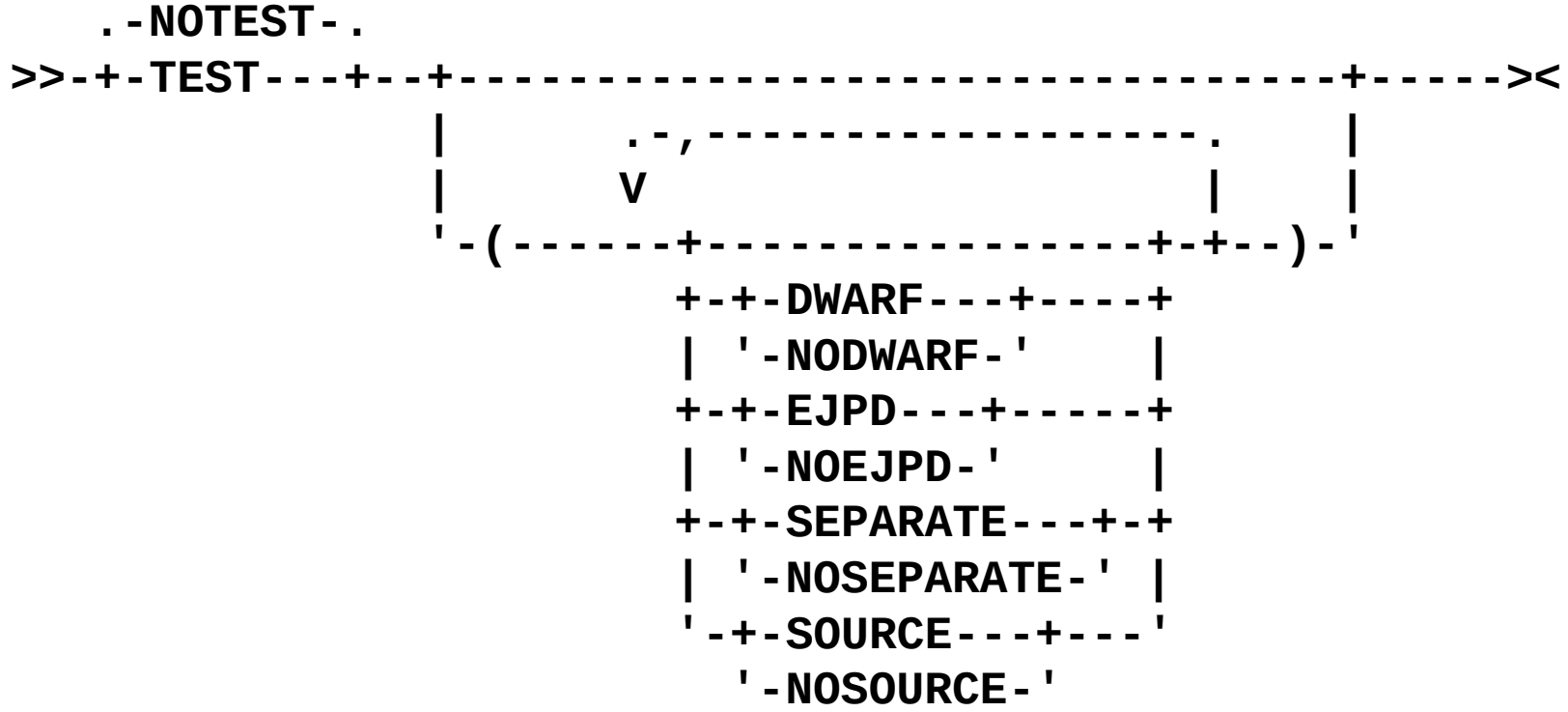
- ‘New’ NUMCHECK option tells the compiler whether to generate extra code to validate data items when they are used as sending data times. For zoned decimal (numeric USAGE DISPLAY) and packed decimal (COMP-3) data items, the compiler generates implicit numeric class tests for each sending field. For binary data items, the compiler generates SIZE ERROR checking to determine whether the data item has more digits than its PICTURE clause allows.
  - **Note:** The ZONECHECK option is deprecated but is tolerated for compatibility, and it is replaced by NUMCHECK(ZON).
- ‘New’ PARMCHECK option tests for subprograms that write beyond the end of WORKING-STORAGE.
  - This option tells the compiler to generate an extra data item following the last item in WORKING-STORAGE that is then used at run time to check whether a called subprogram corrupted data beyond the end of WORKING-STORAGE.
- ‘New’ SSRANGE suboptions allow:
  - MSG/ABD Choose a warning message or an ABEND for SSRANGE conditions
  - ZLEN allows a reference modification of zero length to proceed without a message or abend
- These new options and suboptions were added to help with migration .

New and changed COBOL options

- New combinations of suboptions are supported in both the TEST and NOTEST compiler options, including:
  - TEST(NODWARF)
    - Can help 3rd-party vendor debugging experience without larger object programs
  - TEST(SEPARATE)
    - Helps customers who deploy copies of the same programs to many systems, they can share a separate dataset with one copy of debugging information for all copies – less DASD usage
    - DD SYSDEBUG is back!
  - NOTEST(DWARF)
    - For customers who want full optimization with some debugging information for Fault Analyzer or CEEDUMP (IBM z/OS Debugger, formerly Debug Tool, cannot debug programs compiled with NOTEST)

New and improved TEST option

- Use TEST to produce object code that enables debugging with problem determination tools such as IBM z/OS Debugger and Fault Analyzer.
- Syntax:



- Default: NOTEST(NODWARF, NOSOURCE, NOSEPARATE)
- Suboption defaults:
  - NODWARF, NOSOURCE, NOSEPARATE when NOTEST is specified with no suboptions
  - NOEJPD, DWARF, SOURCE, NOSEPARATE when TEST is specified with no suboptions

New and improved TEST option - review

- Abbreviations: None
- Suboption abbreviation are:
  - NOSO | SO for SOURCE | NOSOURCE
  - NOSEP | SEP for SEPARATE | NOSEPARATE
- DWARF | NODWARF

  If TEST(DWARF) is in effect, complete DWARF diagnostic information is included in the object program, or a separate debug file, when the SEPARATE suboption is in effect. This option enables the best usability for application failure analysis tools, such as CEEDUMP and IBM®Fault Analyzer. When NOTEST(DWARF) is in effect, the debugging information is a subset of the DWARF information that is available with TEST. The DWARF diagnostic information that is produced when NOTEST(DWARF) is in effect, cannot be used with IBM z/OS Debugger.
  If NODWARF is in effect, DWARF diagnostic information is not included in the object program, or written to a separate debug file.

  Notes:

  •SOURCE and SEPARATE are not allowed with NODWARF.
  •If you specify the DWARF suboption of TEST or NOTEST, you must set the CODEPAGE option to the CCSID that is used for the COBOL source program.

New and improved TEST option - review

- EJPD | NOEJPD
  - EJPD and NOEJPD control enablement of the IBM z/OS Debugger commands JUMPTO and GOTO in production debugging sessions. EJPD and NOEJPD only take effect if you specify the TEST option and a non-zero OPTIMIZE level (OPTIMIZE(1) or OPTIMIZE(2)).
  - If you specify TEST(EJPD) and a non-zero OPTIMIZE level:
    - The JUMPTO and GOTO commands are enabled.
    - The amount of program optimization is reduced. Optimization is done within statements, but most optimizations do not cross statement boundaries.
  - If you specify TEST(NOEJPD) and a non-zero OPTMIZE level:
    - The JUMPTO and GOTO commands are not enabled, but you can use JUMPTO and GOTO if you use the SET WARNING OFF IBM z/OS Debuger command. In this scenario, JUMPTO and GOTO will have unpredictable results.
    - The normal amount of program optimization is done.
  - **Note:**
    - EJPD is not allowed with NOTEST.

New and improved TEST option - review

- SOURCE | NOSOURCE
  - If you specify SOURCE, the DWARF debugging information generated by the compiler includes the expanded source
  - **Note:** SOURCE is not allowed if NODWARF is specified.
  - If you specify NOSOURCE, the generated DWARF debugging information does not include the expanded source. You will not be able to debug using the IBM z/OS Debugger with TEST(NOSOURCE).

- SEPARATE | NOSEPARATE
  - Specify SEPARATE to control program object size on disk while retaining debugging capability. Generated DWARF debugging information is written to a separate SYSDEBUG file instead of to the object program.
  - **Note:** SEPARATE is not allowed if NODWARF is specified.
  - Specify NOSEPARATE to include generated DWARF debugging information in the object program.
    (NOSEPARATE does not affect the size of the loaded program object)

Improved compiler listings: usability!

- Improved compiler listings with compiler messages at the end of the listing as in previous releases of the compiler
  - Finally!
  - Much improved integration between compiler Front End and Back End

```
 000011                    GOBACK.
PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.1.0 S170404       MSGS      Date 07/27/2017  Time 19:14:26   Page     4
An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

 Defined   Cross-reference of data names   References

       6   W. . . . . . . . . . . . . . .  9
       7   X. . . . . . . . . . . . . . .  M9
PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.1.0 S170404       MSGS      Date 07/27/2017  Time 19:14:26   Page     5
 Defined   Cross-reference of programs     References

       2   MSGS
PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.1.0 S170404       MSGS      Date 07/27/2017  Time 19:14:26   Page     6
LineID  Message code  Message text

     9  IGYPA3228-W   High order digit positions in the sender may be truncated in the move to receiver "X (NUMERIC INTEGER)".
Messages    Total    Informational    Warning    Error    Severe    Terminating
Printed:        1                         1
* Statistics for COBOL program MSGS:
*    Source records = 11
*    Data Division statements = 2
*    Procedure Division statements = 2
*    Generated COBOL statements = 0
*    Program complexity factor = 2
End of compilation 1,  program MSGS,  highest severity 4.
PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.1.0 S170404       MSGS      Date 07/27/2017  Time 19:14:26   Page     7
 000002:          PROGRAM-ID. MSGS.
    000000                    000002           PROC    MSGS
    000000  47F0 F014         000002           BC      R15,20(,R15)          #  Skip over constant area
    000004  01C3 C5C5         000002           DC      X'01C3C5C5'           #  Eyecatcher: CEE
    000008  0000 01F0         000002           DC      X'000001F0'           #  Stack Size
    00000C  0000 04F0         000002           DC      X'000004F0'           #  Offset to PPA1
    000010  47F0 F001         000002           BC      R15,1(,R15)           #  Wrong Entry Point: cause exception
    000014                    000002  L0130:    EQU     *
    000014  90EC D00C         000002           STM     R14,R12,12(,R13)      #  Save GPRs Used
    000018  41A0 F024         000002           LA      R10,36(,R15)          #  Args for boot strap routine
    00001C  98EF F034         000002           LM      R14,R15,52(,R15)      #
```

Not really the end at all!

Improved compiler listings: usability!

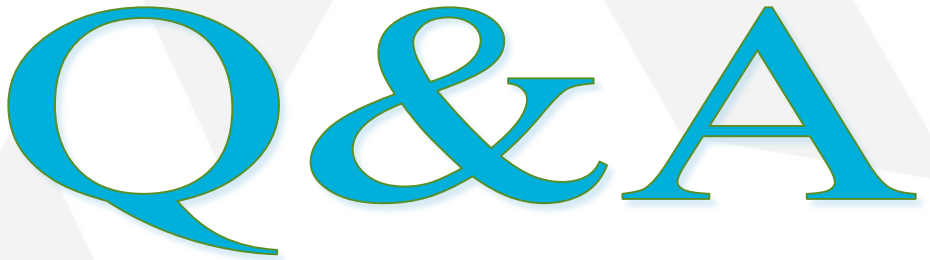
```
          C              4       __CAACRENT
         80              4       _@MYARGS
         84              4       _@CAA
         CC              4       _CACHED_$STATIC


                * * * * *   E N D     O F     A U T O M A T I C     M A P   * * * * *


          Compiler back end level: tr_r16_cobol_20170720_141461_sR_MIG13Eee3XrIDKtOxIA


          * * * * *   E N D   O F   C O M P I L A T I O N   * * * * *

PP 5655-EC6 IBM Enterprise COBOL for z/OS  6.2.0 P170724       MSGS      Date 07/27/2017  Time 19:15:03   Page    17
LineID  Message code  Message text

     9  IGYPA3228-W   High order digit positions in the sender may be truncated in the move to receiver "X (NUMERIC INTEGER)".
Messages    Total    Informational    Warning    Error    Severe    Terminating
Printed:        1                         1
* Statistics for COBOL program MSGS:
*    Source records = 11
*    Data Division statements = 2
*    Procedure Division statements = 2
*    Generated COBOL statements = 0
*    Program complexity factor = 2
End of compilation 1,  program MSGS,  highest severity 4.
Return code 4
```

REALLY the end now!

What else?

- **Improved interfaces to other licensed programs and tools**
  - Addition of MD5 signature to program objects and debug data to allow matching of debug data with executables even if a program is recompiled.
  - Three new fields at the end of PPA4:
    - Offset of the first user-defined data item in WORKING-STORAGE
    - Total length of user-defined data items in WORKING-STORAGE
    - Bit to indicate whether there are EXTERNAL data items
- **Compile-time and runtime performance improvements**
  - General compile-time performance improvements
    - With OPTIMIZE(1) and OPTIMIZE(2)
    - Up to 20% improvement!
  - General batch runtime performance improvements
  - General online transaction runtime performance improvements
- **Usability enhancements in the z/OS UNIX System Services environment**
  - We added help information for the cob2 compiler invocation command.

Questions?



SHARE
EDUCATE ▸ NETWORK ▸ INFLUENCE

Thank You!