Using zCOBOL COMPUTE statement to generate z390 assembler complex arithmetic code

Don Higgins

don@higgins.net

May 22, 2012

You can write a zcobol subroutine written in the COBOL language which uses COMPUTE statements to calculate complex arithmetic expressions when called from z390 mainframe assembler programs. But why would you want to:

1. A single COBOL COMPUTE statement in the example below generates over 300 mainframe assembler instructions required to add 15 different arithmetic fields together.
2. Using COMPUTE allows developer to focus on correctly defining field types, implied decimals, and order of computations using parenthesis without getting lost in the details of coding 300 assembler instructions many of which the developer may not be all that familiar with.
3. Using zcobol and z390 is a great way to learn about the 15 different numeric data field types supported by mainframe assembler, the benefits and limitations of each type, and the associated instructions. The relatively new Decimal Floating Point (DFP) support may be of particular interest as it supports up to 34 decimal digit precision without some of the rounding problems associated with using hexadecimal or binary floating point.

The zcobol language supports the following 15 different data type fields which may be passed to and from a calling mainframe assembler program and which may be used in COMPUTE statements:

1. FLOAT-HEX-7 COMP-1 - EH HFP short - 7 digits
2. FLOAT-HEX-15 COMP-2 - DH HFP long - 15 digits
3. FLOAT-HEX-30 - LH HFP extended - 30 digits
4. FLOAT-BINARY-7 - EB BFP short - 7 digits
5. FLOAT-BINARY-16 - DB BFP long - 16 digits
6. FLOAT-BINARY-34 - LB BFP extended - 34 digits
7. FLOAT-DECIMAL-7 FLOAT-SHORT - ED DFP short - 7 digits
8. FLOAT-DECIMAL-16 FLOAT-LONG - DD DFP long - 16 digits
9. FLOAT-DECIMAL-34 FLOAT-EXTENDED - LD DFP extended - 34 digits
10. COMP S9(4) - H 16 bit binary – 4 digits
11. COMP S9(9) - F 32 bit binary – 9 digits
12. COMP S9(18) - G 64 bit binary – 18 digits
13. COMP S9(39) - Q 128 bit binary – 39 digits
14. COMP-3 S9(31) - P (3) Packed decimal - 31 digits
15. S9(31) Z (3) Zoned Decimal - 31 digits

The non-floating point data types 10-15 above can each have implied decimal places up to the number of decimal digits each type supports. For example a COBOL working storage item 77 F1 COMP PIC S9(7)V99 defines a signed full word binary integer field with 2 implied decimal places named F1. A COBOL COMPUTE statement of the form COMPUTE F1 = 1.5 would store the binary integer value of 150 (hex x'00000096') in the 4 byte binary field named F1.

If you are interested in learning how you can actually code and test this method out on your own Windows, Linux, or Apple OSX personal computer, read on. First you will need to install a version of J2SE Java version 1.6 or later which can be downloaded for free from the following site:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

You will also need to download the latest full release of the z390 Portable Mainframe Assembler which includes zcobol Portable Mainframe COBOL compiler. z390 and zcobol are open source products which can be downloaded for free from the following site:

www.z390.org

This paper was written using examples run on J2SE 1.7.0 and z390 v1.5.06. All the source code files and generated files for this demo are included in the z390 v1.5.06 release and are also available for download from here:

http://www.z390.org/zcobol/demo/callcomp/CALLCOMP.ZIP

The following 34 line CALLCOMP.MLC z390 mainframe assembler source program calls statically linked subroutine COMPSUM to calculate the sum of 15 fields and return the sum in a packed decimal field which is then edited and displayed using write to operator macro:

```
         TITLE 'CALLCOMP - CALL ZCOBOL COMPUTE TO ADD 15 FIELDS'
* AUTHOR. DON HIGGINS.
* DATE.   05/12/12.
* REMARKS. CALL ZCOBOL COMPSUM.CBL TO ADD 15 DIFFERENT FIELD TYPES.
CALLCOMP SUBENTRY
         CALL COMPSUM,(ZDATA)
         ED   DSUM,PSUM
         WTO  MF=(E,MSG)
         SUBEXIT
         PRINT DATA
         LTORG
MSG      DC   AL2(MSGEND-*,0)
         DC   C'COMPUTE TOTAL='
DSUM     DS   0CL(MSGEND-*)
         DC   C' ',X'20202021',C'.',X'20'  ZZZ9.9
MSGEND   EQU  *
ZDATA    DS   0D
EH1      DC   EHL4'1.5'
DH1      DC   DHL8'1.5'
LH1      DC   LHL16'1.5'
EB1      DC   EBL4'1.5'
DB1      DC   DBL8'1.5'
LB1      DC   LBL16'1.5'
ED1      DC   EDL4'1.5'
DD1      DC   DDL8'1.5'
LD1      DC   LDL16'1.5'
```

```
H1      DC   HL2'15'          3V1
F1      DC   FL4'150'         7V2
G1      DC   FL8'1500'       15V3
Q1      DC   FL16'15000'     25V4
P1      DC   PL16'150000'    26V5
Z1      DC   ZL31'1500000'   25V6
PSUM    DC   P'00000'         4V1
        END
```

The following 32 line COMPSUM.CBL zcobol mainframe COBOL program uses a COMPUTE statement to add the 15 fields defined in linkage section and store the result in a packed decimal field in the same linkage section:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    COMPSUM
       AUTHOR.        DON HIGGINS.
       DATE-WRITTEN.  05/12/12.
      * SUBROUTINE CALLED TO ADD 15 DATA FIELDS AND RETURN SUM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       LINKAGE SECTION.
       01 ZDATA.
         05 EH1 FLOAT-HEX-7.
         05 DH1 FLOAT-HEX-15.
         05 LH1 FLOAT-HEX-30.
         05 EB1 FLOAT-BINARY-7.
         05 DB1 FLOAT-BINARY-16.
         05 LB1 FLOAT-BINARY-34.
         05 ED1 FLOAT-DECIMAL-7.
         05 DD1 FLOAT-DECIMAL-16.
         05 LD1 FLOAT-DECIMAL-34.
         05 H1      COMP PIC S9(3)V9.
         05 F1      COMP PIC S9(7)V99.
         05 G1      COMP PIC S9(15)V999.
         05 Q1      COMP PIC S9(35)V9999.
         05 P1      COMP-3 PIC S9(26)V9(5).
         05 Z1             PIC S9(25)V9(6).
         05 PSUM    COMP-3 PIC S9(4)V9.
       PROCEDURE DIVISION USING ZDATA.
           COMPUTE PSUM = EH1+DH1+LH1
                   +EB1+DB1+LB1
                   +ED1+DD1+LD1
                   +H1+F1+G1
                   +Q1+P1+Z1         .
           GOBACK.
```

From the z390 command line or GUI interface, the following command can be used to compile, statically link, and execute these two source programs:

zcobol\demo\callcomp\CLG.BAT

The above batch command performs the following 2 steps:

1. ZC390C COMPSUM - this command compiles COMPSUM.CBL into COMPSUM.MLC assembler source code with data labels and then assembles COMPSUM.MLC into relocatable object code in COMPSUM.OBJ. In addition a COMPSUM.BAL basic assembler source file, a COMPSUL.PRN assembler listing file, and a COMPSUM.ERR systerm log file with any assembler errors plus all MNOTE statements are generated.
2. ASMLG CALLCOMP – this command assembles CALLCOMP.MLC into relocatable object code file CALLCOMP.OBJ, links the relocatable object code including called module COMPSUM into executable load module CALLCOMP.390, and then executes the load module producing execution log file CALLCOMP.LOG which contains write to operator messages which are also displayed on console.

The execution log file CALLCOMP.LOG after executing the above 2 commands contains the follow:

```
13:13:25 CALLCOMP  EZ390 START USING z390 V1.5.06rx4 ON J2SE 1.7.0 05/12/12
EZ390I Copyright 2011 Automated Software Tools Corporation
EZ390I z390 is licensed under GNU General Public License
EZ390I program = W:\work\z390\CALLCOMP.390
EZ390I options = W:\work\z390\z390.OPT=(time(60))
COMPUTE TOTAL=   22.5
EZ390I instructions/sec     = 10406
EZ390I total errors         = 0
13:13:25 CALLCOMP  EZ390 ENDED   RC= 0 SEC= 0 MEM(MB)= 12  IO=189 INS=333
```

The above execution log shows that the z390 emulator executed 333 mainframe assembler instructions. To add the 15 data fields and display the edited sum. To see the instructions generated by the zcobol compiler for the COMPUTE statement in COMPSUM.CBL were, there are three options available:

1. Set &DEBUG option on in zcobol\ZC_CALC.MAC to generate MNOTE for each COBOL command generated by each COMPUTE statement. The MNOTE statements generated during compilations can be viewed on COMPSUM.ERR file.
2. View the generated basic assembler file COMPSUM.BAL after compilation.
3. Add z390 execution trace option on the ASMLG CALLCOMP TRACE(E) command to generate trace file CALLCOMP.TRE which shows each of the 333 instructions executed along with the associated instruction operand register and/or storage value plus condition code result for each instruction.

For the first option, the macro variable &DEBUG is set to 1 in the zcobol compiler macro zcobol\ZC_CALC.MAC, then the z390 assembly process used by the zcobol compiler, will generate MNOTE statements for each of the COBOL MOVE, ADD, SUBTRACT, MULTIPLY, and DIVIDE commands generated to perform each COMPUTE statement. The first MNOTE for each COMPUTE statement shows COMPUTE parameters. The following indented MNOTE statements show operation, target field name, target type, target length, target decimal places, source field name, source type, source length, and source decimal places. The COMPSUM.ERR file generated by the above compile contains the following:

```
 13:13:19 COMPSUM   ZC390 START USING z390 V1.5.06rx4 ON J2SE 1.7.0 05/12/12
13:13:19 COMPSUM   ZC390 ENDED   RC= 0 SEC= 0 MEM(MB)=  7 IO=2
13:13:19 COMPSUM   MZ390 START USING z390 V1.5.06rx4 ON J2SE 1.7.0 05/12/12
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC
PARMS=PSUM,=,EH1,+,DH1,+,LH1,+,EB1,+,DB1,+,LB1,+,ED1,+,DD1,+,LD1,+,H1,+,F1,+,G1,+,Q1,+,P1,+,Z1'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1,9,16,0 EH1,1,4,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1,9,16,0 DH1,2,8,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2,9,16,0 TMP-1,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2,9,16,0 LH1,3,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1,9,16,0 TMP-2,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1,9,16,0 EB1,4,4,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2,9,16,0 TMP-1,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2,9,16,0 DB1,5,8,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1,9,16,0 TMP-2,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1,9,16,0 LB1,6,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2,9,16,0 TMP-1,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2,9,16,0 ED1,7,4,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1,9,16,0 TMP-2,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1,9,16,0 DD1,8,8,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2,9,16,0 TMP-1,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2,9,16,0 LD1,9,16,0'
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1, 9,16,0 TMP-2, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1, 9,16,0 H1, H, 2, 1’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2, 9,16,0 TMP-1, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2, 9,16,0 F1, F, 4, 2’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1, 9,16,0 TMP-2, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1, 9,16,0 G1, G, 8, 3’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-2, 9,16,0 TMP-1, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-2, 9,16,0 Q1, Q, 16, 4’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE TMP-1, 9,16,0 TMP-2, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD TMP-1, 9,16,0 P1, P, 16, 5’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   MOVE PSUM, P, 3, 1 TMP-1, 9, 16, 0’
13:13:21 COMPSUM   AZ390 MNOTE 'ZC_CALC   ADD PSUM, P, 3, 1 Z1, Z, 31, 6’
13:13:22 COMPSUM   MZ390 ENDED   RC= 0 SEC= 2 MEM (MB) =113 IO=35885
```

The above MNOTE’s generated by the ZC_CALC structured conditional macro assembler module illustrate how the COMPUTE statement expression is parsed using a Backus Normal Form (BNF) operation stack and a variable stack and results in generating sequential calls to the basic COBOL MOVE, ADD, SUBTRACT, MULTIPLY, and DIVIDE modules which in turn generate assembler instructions to perform the requested operations including any scaling required to adjust for implied decimal points. The ZC_CALC module uses dynamic allocation and de-allocation of temporary storage locations required based on the implied or explicit order required based on the operation precedence and explicit parenthesis. A z390 pending RPI 1214 has been added for future optimization of ZC_CALC to reduce unnecessary temporary storage and further optimize the instructions used while maintaining required precision.

The second option to view the specific generated z390 basic mainframe assembler source code for the zcobol source program can be viewed to see the generated mainframe instructions required to add the

15 different type fields. The instructions used include HFP, BFP, and DFP floating point plus 64 bit instructions, packed decimal, and zoned decimal. Conversion instructions such as PFPO are also used. For data type Q 128 bit integers several zcobol runtime routines are called to perform 128 bit multiplication and division using 64 bit interger instructions. The MNOTE's generated by the &DEBUG option in ZC_CALC appear in the basic assembler source and can be used to search for the generated code associated with a specific field name. The 300+ assembler instructions generated for the single COMPUTE statement can be found in the following file:

http://www.z390.org/zcobol/demo/callcomp/CALLCOMP.ZIP (file COMPSUM.BAL)

For option 3, assemble, link, and execute with TRACE (E) option which produces the following z390 execution trace showing all 333 instructions executed along with associated register and storage input field values for the execution of each instruction:

http://www.z390.org/zcobol/demo/callcomp/CALLCOMP.ZIP (file COMPSUM.TRE)

For reference there is a batch command to compile and execute this example in the zip file here:

http://www.z390.org/zcobol/demo/callcomp/CALLCOMP.ZIP (file CLG.BAT)

Also for reference here is link to the IBM z/OS Principles of Operations manual describing all the assembler instructions supported by the latest IBM z196 mainframe processor:

http://publibfi.boulder.ibm.com/epubs/pdf/dz9zr008.pdf

I hope you find this paper useful. Any comments may be sent to don@higgins.net.