zCOBOL

System Programmer’s Guide

v1.5.00

Automated Software Tools Corporation.

zc390 Translator

The zc390 translator is a java regular expression based parser which reads COBOL source program and generates HLASM compatible mainframe assembler source program in one pass. Each recognized COBOL verb starts a new assembler macro call statement with all the parameters up to the next verb, period, or paragraph label passed as positional parameters. Periods generate a separate macro call to PERIOD to generate end to all the structures in the previous sentence. Paragraph and section labels generate call to LABEL with the name and type of label to generate. All hyphens in names are translated to underscores for HLASM compatibility.

COBOL Language Verb Macros

All the macros for the COBOL language verbs and section headings are stored in the macro library z390\zcobol. These macros parse the parameters, validate them for any syntax errors, and issue calls to generation macros in separate directory as described below. For example, the zcobol\IF.MAC macro generates multiple calls to the generation macros GEN_COMP, GEN_BC GEN_B, and GEN_LABEL. There are no language specific code generation macros in the zCOBOL directory so it is shared across multiple target language environments. All the macros are written in structured form using the z390 ZSTRMAC SPE structured programming extensions such as AIF, AELSEIF, AELSE, AEND, AWHILE, etc. As a result there are no explicit AGO or AIF labels in these macros.

COMPUTE Statement Example

The COBOL compute statement now supported in v1.5.00d is a good example to study to understand how the zCOBOL compiler works. The steps followed to compile the following MOVE and COMPUTE statements are as follows:

```
 77      FLT-SRT USAGE FLOAT-SHORT OCCURS 2.
........
MOVE 1.1 TO FLT-SRT(2)
COMPUTE FLT-SRT(2) = FLT-SRT(2)+2.2
```

1 zc390 translator generates the following 2 zCOBOL verb macro call statements
```
    MOVE 1.1,'TO',FLT_SRT,"(',2,')'
    COMPUTE FLT_SRT,'(',2,')',=,FLT_SRT,'(",2,')',+,2.2
```
2 The MOVE macro uses shared copybook routine GET_FIELD_PARM to parse the two fields for MOVE and store resulting field name and symbol table index. For the literal 1.1 the index is 0, for the subscripted field, the name is set to explicit register reference including length 0ffset(length,register) and the code is generated to set the register to address of the subscripted field.

3 The MOVE macro next issues call to GEN_MOVE with the source and target field names and system table indexes.

4 The GEN_MOVE macro checks the type of each field and generates appropriate code to move value from source to target field. In this case it uses LARL to set register to address of DFP short value of 1.1 in literal table and then generates MVC to move the literal to the target subscripted field.

5 The COMPUTE uses GET_FIELD_PARM to obtain name and index of target field and then extracts parms in expression following the = and then calls ZC_CALC macro to generate code for expression and store result in specified target field. This macro can be used by IF and other verb macros to calculate expression for loop etc.

6 The ZC_CALC macro parsers the expression parameters into Backus Normal Form using two stacks. One stack has the operators in expression and the other has the field parm index pointers. Following the rules of precedence, the operators and associated parameter pointers are removed from the stacks and stored sequentially in an operation table containing the operators, 2 operands, and the target field for each operation. Temporary storage fields are represented using negative indexes instead of position and a table of temporary fields created along with their type is maintained. A queue of free temporary fields is maintained and once a temporary field has been used in an operation, that temporary field is on the free queue for reuse rather than allocating a new temporary storage field. Once the expression has been parsed and all the operation table entries have been generated, the last target field is replaced with the result field passed to ZC_CALC and then the operation table is scanned and the generation macros for each operation are called to generate code to perform the operation. Just prior to generating code for an operation, the two input parameter types are used to determine the required type of result to minimize any loss of precision during the calculations. A call to GEN_MOVE is made to move the first operand field to the target field prior to performing add, subtract, etc. on the target field for operation. If the first operand is the same as the target field, the move can be omitted but that is not always possible to determine in the case of subscripting and indexing where different variables may just happen to have the same value. The called generation macros GEN_ADD, GEN_SUB, GEN_MPY, and GEN_DIV check the field types and perform the necessary conversion when types do not match.

The generated zCOBOL HLASM assembler code for these 2 COBOL statements is as follows:

```
0000DC (1/99)652 * MOVE 1.1 TO FLT-SRT(2)
0000DC (1/100)653 MOVE 1.1,TO,FLT_SRT,'(',2,')'
0000DC C02900000004 000004 (29/28)660+ IILF R2,(2-1)*4 (LIT-1)*LEN
0000E2 4122D058 000220 (29/29)661+ LA R2,FLT_SRT(R2)
0000E6 C0100000006D 0001C0 (30/193)664+ LARL R1,=ED'1.1'
0000EC D20320001000 (30/194)665+ MVC 0(4,R2),0(R1)
0000F2 (1/101)668 * COMPUTE FLT-SRT(2) = FLT-SRT(2)+2.2
0000F2 (1/102)669 COMPUTE FLT_SRT,'(',2,')',=,FLT_SRT,'(',2,')',+,2.2
0000F2 C0F900000004 000004 (29/28)677+ IILF R15,(2-1)*4 (LIT-1)*LEN
0000F8 41FFD058 000220 (29/29)678+ LA R15,FLT_SRT(R15)
0000FC C02900000004 000004 (29/28)688+ IILF R2,(2-1)*4 (LIT-1)*LEN
000102 4122D058 000220 (29/29)689+ LA R2,FLT_SRT(R2)
000106 D203F0002000 (30/64)694+ MVC 0(4,R15),0(R2)
00010C 780F0000 (34/132)697+ LE F0,0(R15)
000110 B3D40000 (34/133)698+ LDETR F0,F0,0
000114 C0E000000052 0001B8 (34/134)699+ LARL R14,=DD'2.2'
00011A 681E0000 (34/135)700+ LD F1,0(R14)
00011E B3D21000 (34/136)701+ ADTR F0,F0,F1
000122 B3D50000 (34/137)702+ LEDTR F0,0,F0,0
000126 700F0000 (34/138)703+ STE F0,0(R15)
```

See the results for this calculation in the regression test zcobol\test\TESTSIX1.CBL. Note that using DFP the result is exactly 3.3 which is not the case when using HFP COMP-1 floating point due to conversion to base 2 versus base 10. The zcobol\ZC_CALC.MAC macro is written in ZSTRMAC structured conditional macro assembler consisting of about 850 lines of code with no AGO statements.

zCOBOL Target Source Language Generation Macros

All the target source language generation macros called by the COBOL verb macros in z390\zcobol are stored in the following directories by target language:

z390\zcobol\z390	Generate HLASM compatible mainframe assembler source program
z390\zcobol\java	Generate J2SE java compatible source program
z390\zcobol\vce	Generate MS Visual Express C compatible source program
z390\zcobol\i586	Generate HLA/MASM Intel assembler compatible source program

Current only the z390 HLASM compatible source generation macros are being fully developed along with the required runtime support functions stored in the zcobol\z390\ZC390LIB.390 dynamically loaded runtime module. However the zCOBOL demos include a hello world COBOL program which can be compiled and executed in each of the target environments form the same zcobol\demo\HELLO.CBL source program. The following commands generate the corresponding source language equivalent and executable:

Command	Generated Source Code Target	Generated Executable Code
ZC390CLG zcobol\demo\HELLO	zcobol\demo\HELLO.MLC/BAL	zcobol\demo\HELLO.390 requires z390 and J2SE on Windows/Linux
ZCJAVCLG zcobol\demo\HELLO	zcobol\demo\HELLO.java	zcobol\demo\HELLO.class requires J2SE on Windows/Linux
ZCVCECLG zcobol\demo\HELLO	zcobol\demo\HELLO.ccp	zcobol\demo\HELLO.exe requires MS VCE runtime on Windows
ZC586CLG zcobol\demo\HELLO	zcobol\demo\HELLO.HLA/ASM	zcobol\demo\HELLO.exe requires HLA, MASM, and MS VCE runtime on Windows

If you are interested in joining in the open source zCOBOL development effort in any of the 4 target language environments or want to add another target language environment, join the zcobol development email discussion group and make your interests known. Melvyn Maltz is currently developing additional EXEC CICS support for zCOBOL programs.

ZC390LIB Runtime Library

The z390\zcobol\z390 code generation macro directory also contains all the source code and the ZC390CVT.CPY copybook required to build the z390\linklib\ZC390LIB.390 runtime load module which is dynamically loaded by all generated z390 zCOBOL programs. This module contains the following components:

ZC390LIB.MLC	Contains ZC390LIB CSECT and COPY ZC390CVT to include all object modules following the CVT at the beginning
ZC390NUC.MLC	Included module with system function routines such as CALL, GOBACK, STOPRUN, PERFORM, and PMCHECK to check for end of current performed paragraph or section
ABORT.MLC	Contains module called to abort execution with reason code
ACCEPT.MLC	Contains support for ACCEPT date, time, day of week
DISPLAY.MLC	display any type field or literal
INSPECT.MLC	Inspect field tallying, replacing, or transforming.

The ZC390CVT.CPY copybook is used in every zCOBOL generated program to define the DSECT addressed by register 9. The same copybook is also used in ZC390LIB.MLC to generate the CVT at the beginning of the ZC390LIB.390 runtime load module with addresses of all the entries followed by work areas used by the code generation macros.

Base Free Code Generation

The zCOBOL code generation macros in zcobol\z390 generate base free code for the procedure division using relative instructions for both branch addressing and for literal addressing as required. The only address constants generated in zCOBOL programs are for statically linked CALL's to other zCOBOL or assembler programs. The only limit on the combined size of working storage and the procedure division is 16 MB. In order to use relative addressing for literals, all odd length literals are padded to even lengths. The LARL instruction is used to set address of data field or literal field as required for use in following RX type instructions. To address working storage and linkage section data fields, conventional base registers are dynamically allocated as required for use in RX type instructions. Since R13 always points to the beginning of working-storage, no dynamic base registers are required for access to data items in the first 4k of working storage.

zCOBOL EXEC CICS Support

When the option CICS is specified on the command line for ZC390C, ZC390CL, or ZC390CLG, then the zcobol\ZCOBOL. MAC global option &ZC_CICS is set on and the following changes in code generation are made:

1 The CICS option will generate call to DFHEIENT to initialize CICS prior to executing user code starting at the first program CSECT.
2 A DFHEISTG DSECT is generated at the beginning of working-storage instead of WSLOC LOCTR and warnings are generated for any data VALUE clauses defined in working-storage section.

zCOBOL Data Types

USAGE	PICTURE	Z390 Assembler Type	Description
COMP	S9(4)	H	16 bit binary
COMP	S9(9)	F	32 bit binary
COMP	S9(18)	G	64 bit binary
COMP	S9(39)	Q	128 bit binary
FLOAT-HEX-7	COMP-1	EH	HFP short 7 digits
FLOAT-HEX-15	COMP-2	DH	HFP long - 15 digits
FLOAT-HEX-30		LH	HFP extended - 30 digits
FLOAT-BINARY-7		EB	BFP short 7 digits
FLOAT-BINARY-16		DB	BFP long - 16 digits
FLOAT-BINARY-34		LB	BFP extended - 34 digits
FLOAT-DECIMAL-7	FLOAT-SHORT	EB	DFP short 7 digits
FLOAT-DECIMAL-16	FLOAT-LONG	DB	DFP long - 16 digits
FLOAT-DECIMAL-34	FLOAT-EXTENDED	LB	DFP extended - 34 digits
COMP-3	S9(31)	P (3)	Packed decimal up to 31 digits with option EXTEND
	S9(31)	Z (3)	Zoned Decimal up to 31 digits with option EXTEND (uses PD support)

	X	X	Characters
FLOAT-SHORT		EH,EB,ED	Use option FLOAT(HFP/BFP/DFP)
FLOAT-LOG		DH,DB,DD	Use option FLOAT(HFP/BFP/DFP)
FLOAT-EXTENDED		LH,LB,LD	Use option FLOAT(HFP/BFP/DFP)
POINTER PROCEDUREPOINTER		A	31 bit binary

1 The zCOBOL option FLOAT(HEX/BINARY/DECIMAL) can be used to change the default from DECIMAL to HEX or BINARY for the generic types FLOAT-SHORT, FLOAT-LONG, and FLOAT-EXTENDED.

2 COMP-3 packed and also zoned decimal are limited to 18 digits per COBOL standard unless option EXTEND is set allowing up to 31 digits for both packed decimal and zoned decimal fields.z390 and zCOBOL options include the following:

Command Line options for zCOBOL Compiler

To turn off an option that is on, prefix the option name with NO on command line or in OPT options file.

Option	Default	Description
@file	NO	Retrieve additional options from free form text file with default suffix OPT. Options can be specified delimited by spaces on as many lines as requires. All characters on a line following * are ignored as comments. The @file option can be nested. The default path is the program path.
CICS	NO	Parse COBOL EXEC CICS commands into z390 EXEC CICS compatible macro calls and also rename working storage to DFHEISTG.
COMMENT	YES	Generate MLC comments showing original COBOL statement preceding each macro call statement.
EXTEND	YES	Support up to 31 digits for DISPLAY (Z) and COMP-3 (P) type data items rather than limiting precision to ANSI 1985 standard of 18.
FLOAT (DECIMAL)	YES	Set type of floating point for usage FLOAT-SHORT, FLOAT-LONG, and FLOAT-EXTENDED. The choices are FLOAT(HEX) for Hexadecimal Floating Point (HFP) like COMP-1 and COMP-2, FLOAT(BINARY) for Binary Floating Point (BFP), or the default FLOAT(DECIMAL) for Decimal Floating Point (DFP).
R64	YES	Generate 64 bit instructions for the 16 GPR registers where appropriate. NOR64 restricts code generation to only use lower 32 bits of 16 GPR registers as required by z/VSE and some other operating environments. (Note option TRUNC and NOR64 results in use of DXR instead of DGR which is more efficient.)
TRACE	NO	Generate WTO display of paragraph name at entry to each new paragraph in NO procedure division. This provides high level trace as opposed to using the z390 TRACE(E) option which generates instruction level trace.
TRUNC	NO	Truncate binary data types F, G, and H to specified number of digits in PICTURE.
WARN	YES	Generate level 4 MNOTE warnings from zCOBOL macros

- Options are passed to the zCOBOL macro stage via CBL macro call with the options defined as positional parameters

zCOBOL File Types

TYPE	Format	File Description	File or Report Format Description
CBL	ASCII	COBOL source program	1-6 sequence #, 7 Comment if not space, 8-11 area A, 12-72 area B
MLC	ASCII	Macro assembler source program generated by phase 1 of the zCOBOL compiler which uses zcobol.class regular expression based parser in z390.jar to read CBL source file and create MLC source file in one pass.	Macro call. for each COBOL statement starting in area A and for each COBOL verb found in area B. Working storage data items are mapped to WS macro call with level as first parameter. Each macro call name is followed by positional parameters found following verb up to next verb or period. Periods are mapped to PERIOD macro call. Parameters of the form keyword(..) are passed as single parameter. Other (and) are passed as separate parameter in quotes.
BAL	ASCII	HLASM compatible source code generated by phase 2 of the zCOBOL compiler when using ZC390C or ZC390CLG commands.	HLASM compatible source statements generated by the zcobol+zcobol\z390 macros during expansion of the generated MLC file.
JAVA	ASCII	J2SE Java compatible source program file generated by phase 2 of the zCOBOL compiler when using ZCJAVC or ZCJAVCLG commands.	Java source statements generated by the zcobol+zcobol\java macros during expansion of the generated MLC file.
CPP	ASCII	MS Visual Express C compatible source program file generated by phase 2 of the zCOBOL compiler when using ZCVCEC or ZCVCECLG commands.	MS Visual Express C compatible source statements generated by the zcobol+zcobol\vce macros during expansion of the generated MLC file.
HLA	ASCII	HLA/MASM compatible source program file generated by phase 2 of the zCOBOL compiler when using ZC586C or ZC586CLG commands.	HLA/MASM compatible source statements generated by the zcobol+zcobol\i586 macros during expansion of the generated MLC file.

Trademarks

IBM, CICS, HLASM, MVS, OS/390, VSAM, z9, z10, and z/OS are registered trademarks of International Business Machines Corporation

Credits

Author	: Don Higgins
Formatting	: Walter Petras
Z390 version	: