#SHAREorg

SHARE
Technology · Connections · Results

Using the HLASM Macro Facility to Improve Assembler Language Programs

Ed Jaffe
Phoenix Software International

August 9, 2012
Session Number 11179

SHARE in Anaheim 2012

Macro Facility Overview

- The HLASM macro facility is extremely powerful, especially when compared against the preprocessor capabilities offered by other languages.
- Leveraging this power can simplify HLASM programs and ease development/maintenance burdens.
- As powerful as the macro facility is, some assembler programmers avoid its use or question its applicability for "ordinary" HLASM programs.
- In my opinion, HLASM programming without using macros is like going back in time half a century.
- Macros should be thought of as a fundamental and necessary part of every assembler language programmer's toolset.

Using Macros

- The main use of macros is to insert assembler language statements into a source program.
- You call a named sequence of statements (the macro definition) by using a macro instruction, or macro call. The assembler replaces the macro call by the statements from the macro definition and inserts them into the source module at the point of call.
- The process of inserting the text of the macro definition is called macro generation or macro expansion. Macro generation occurs during conditional assembly.
- The expanded stream of code then becomes the input for processing at assembly time; that is, the time at which the assembler translates the machine instructions into object code.

SHARE
Technology · Connections · Results

Macro Definition

- The header statement: MACRO
- The macro prototype statement. This defines the name of your macro and the parameters (if any) that it will accept.
- The macro body consisting of statements that are generated when you call the macro; they are usually interspersed with conditional assembly statements or other processing statements including macro comments.
- The trailer statement: MEND

```
        MACRO ,
&Lbl    MVCLR &Fld,&Value
        MVI   &Fld,&Value          Set field to value
        MVC   &Fld+1(L'&Fld.-1),&Fld           (same)
        MEND  ,
```

SHARE
Technology · Connections · Results

Macro Invocation

```
                                      16062          PRINT NOGEN
001B24 9240 B5A8      0005A8          16063          MVCLR WORKD,C' '          Set field to blanks
001B2E 92FF B5C5      0005C5          16066          MVCLR WORKE2,X'FF'        Set field to x'FF'
001B38 924B A7F6      0007F6          16069          MVCLR EMRMSCMD,C'.'       Set field to dots
```

```
                                      16072          PRINT GEN
                                      16074          MVCLR WORKD,C' '          Set field to blanks
001B42 9240 B5A8      0005A8          16075+         MVI   WORKD,C' '          Set field to value
001B46 D206 B5A9 B5A8 0005A9 0005A8   16076+         MVC   WORKD+1(L'WORKD-1),WORKD       (same)
                                      16077          MVCLR WORKE2,X'FF'        Set field to x'FF'
001B4C 92FF B5C5      0005C5          16078+         MVI   WORKE2,X'FF'        Set field to value
001B50 D207 B5C6 B5C5 0005C6 0005C5   16079+         MVC   WORKE2+1(L'WORKE2-1),WORKE2    (same)
                                      16080          MVCLR EMRMSCMD,C'.'       Set field to dots
001B56 924B A7F6      0007F6          16081+         MVI   EMRMSCMD,C'.'       Set field to value
001B5A D27E A7F7 A7F6 0007F7 0007F6   16082+         MVC   EMRMSCMD+1(L'EMRMSCMD-1),EMRMSCMD
                                                +                                      (same)
```

Location of Macro Definitions

- You can define macros in-line in your program or in a macro library.
- In a macro library, the member name must match the name defined in the macro prototype. If you want to define macros this way, be sure to make the macro names conform to member name rules.
  - Example: macro names must be <=8 characters in length for residency in z/OS PDS library.
  - Otherwise macro names may be up to 63 characters long.
- In-line macros can be defined anywhere in your program prior to first reference.
- In-line macros may be defined by another macro or within a member brought into the program via COPY statement.

Macro Processing Statements

- Processing statements are handled during conditional assembly, when macros are expanded, but they are not themselves generated for further processing at assembly time:
  - AEJECT and ASPACE instructions – control listing of macro definition
  - AREAD instructions – read entire source statements into macro SETC symbols
  - AINSERT instructions – generate complete assembly source statements after the macro generator finishes processing
  - MEXIT instructions – exit from macro processing
  - MNOTE instructions – generate a message
  - Inner macro calls
  - Conditional assembly instructions

Conditional Assembly Language

- The conditional assembly language contains most of the features that characterize a programming language. For example, it provides:
  - Variables
  - Data attributes
  - Built-in functions
  - Expression computation
  - Assignment instructions
  - Labels for branching
  - Branching instructions
  - Substring operators that select characters from a string

Conditional Assembly Language

- The conditional assembly language is *not* structured. Its syntax is loosely-based on the original FORTRAN language specification. ☹
- Structured programming constructs are made available for "ordinary" assembler language coding using the Structured Programming Macros in the IBM HLASM Toolkit or similar. Unfortunately, such tools can *not* be used to structure the conditional assembly language.
- FYI: Don Higgins' z390 package provides a structured macro coding facility called ZSTRMAC. It is a pre-processor that emits input to HLASM. I have no personal experience with this package, but it looks interesting.

http://www.z390.org/z390_ZSTRMAC_Structured_Macro_Support.htm

Conditional Assembly Variables (SET Symbols)

- Symbols may be defined with either global or local scope.
  - Symbols with global scope can be shared by macros and open code.
  - Symbols with local scope are used only within the current macro invocation or in open code.
- SETA symbols are arithmetic; SETB symbols are logical (1=TRUE, 0=FALSE); SETC symbols are character strings.
- Symbols can be arrays.
- Substrings of SETC symbols may be processed.
- Symbols may be dynamically created from the values contained within existing symbols. This can be used in all sorts of clever ways. The common usage I've seen is similar to stem variables usage in Rexx.
- Numerous read-only system symbols are available to convey useful information (e.g., the name of the current section is &SYSECT).

SHARE
Technology · Connections · Results

Some Basic SETA Examples

```
&I        SETA  1                 has value 1
&J        SETA  &I+1              has value 2
&K        SETA  &I+&J             has value 3

&I        SETA  C'A'              has value 193 (C'A')

&I        SETA  INDEX('ABC','B')  has value 2

          LCLA  &A(10)            define array w/10 elements
&I        SETA  1                 do i=1 to 10
.LOOP     ANOP  ,                   .
&A(&I)    SETA  &I*&I               a(i) = i**2
&I        SETA  &I+1                .
          AIF   (&I LE 10).LOOP   end
```

Some Basic SETB Examples

```
&I      SETB  1
&J      SETB  0
&K      SETB  (&I AND &J)       has value 0
&L      SETB  (&I OR &J)        has value 1
&M      SETB  (NOT(&I OR &J))   has value 0

&I      SETA  C'A'              has value 193 (C'A')
&J      SETB  (&I LT 100)       has value 0
```

SHARE
Technology · Connections · Results

Some Basic SETC Examples

```
&C1       SETC   'ABC'              has value ‘ABC’

&C1       SETC   (3)'ABC'           has value ‘ABCABCABC’

&C1       SETC   'ABC'(2,1)         has value ‘B’

          LCLC   &C1(10)
&C1(1)    SETC   'ABC'
&C2       SETC   '&C1(1)'           has value ‘ABC’
&C2       SETC   '&C1(1)'(2,1)      has value ‘A’

&C3       SETC   LOWER('A')         has value ‘a’
```

Conditional Assembly Instructions

Instruction	Operation Performed
GBLA, GBLB, GBLC LCLA, LCLB, LCLC	Declaration of variable symbols (global-scope and local-scope SET symbols) and setting of default initial values
SETA, SETB, SETC	Assignment of values to variable symbols (SET symbols)
SETAF, SETCF	External function assignment of values to variable symbols (SET symbols)
ACTR	Setting loop counter
AGO	Unconditional branch
AIF	Logical test and condition branch
ANOP	Placeholder – no-operation.

```
.* Set Long Branch/Jump Instructions
&JL      SETC  'J'                    Set relative branches
&JLLK    SETC  'JAS'                  (same)
         AIF   (&SYSALVL LT 2).BRJU   Branch if short relative
&JL      SETC  'JL'                   Set long relative branches
&JLLK    SETC  'JASL'                 (same)
.BRJU    ANOP  ,
```

Conditional Assembly in Open Code

- While primarily found in macros, conditional assembly statements also work in open code.

```
         IEZDEB ,                      Define DEB DSECT
         AIF (D'DEBNmTrkHi).z17DEB
DEBNmTrkHi EQU DEBBINUM+1              High byte of track count
.z17DEB  ANOP  ,

* Popular Seconds Values
&SecsIn1Min  SETA 1*60                1/60 hour = 1 min * 60 secs
&SecsIn15Min SETA 15*60               1/4 hour = 15 min * 60 secs
&SecsIn30Min SETA 30*60               1/2 hour = 30 min * 60 secs
&SecsIn1Hr   SETA 60*60               1hr = 60 min * 60 secs
&SecsIn4Hrs  SETA 4*60*60             4hrs = 4hr * 60 min * 60 secs
&SecsIn1Day  SETA 24*60*60            1day = 24hr * 60min * 60sec
&SecsIn1Wk   SETA 7*24*60*60          1wk = 7day * 24hr * 60min * 60sec
.
.
.
     LG    R1,0(,R2)                  Load TOD value
     ALG   R1,=FL8S12'&SecsIn1Hr.E6' Add 60 minutes
     STG   R1,0(,R2)                  Update TOD value
```

SHARE
Technology · Connections · Results

Breaching HLASM's Blood-Brain Barrier

- It's no surprise that SET symbols can be used to generate values for ordinary assembler statements. This is what's expected from any pre-compile language.
- A surprising HLASM "feature" (inherited from older IBM assemblers) allows a SET symbol to be set from an already-defined ordinary symbol with an absolute value.
- This unexpected behavior is *extremely* useful!
- (If you don't realize how awesome this is right now, keep pondering and eventually it should become clear…)

```
STRUCT   DC    F'123'
         DC    F'456'
         DC    XL2'00'
STRUCTLN EQU   *-STRUCT
&STRLEN  SETA  STRUCTLN
         MNOTE *,'Length of structure is &STRLEN'
+*,Length of structure is 10
```

Using a Macro as a Service Interface

- This is one of the two most recognized uses of a macro.
- Common services such as STORAGE, OPEN, CLOSE, GET, PUT, etc. are all implemented using macros.
- This design has advantages over the typical positional parameter CALL interfaces used to invoke services in most languages.
  - The actual byte-level interface need not be documented. Therefore, the existing parameter list can be enhanced and/or extended without changing the callers.
  - With typical positional-parameter interfaces, you generally add new parameters at the end, create a new entry point, or pass a parameter structure to which you append your new parameters.

Using a Macro as a Service Interface

- This macro call searches a z/OS log stream for a specific time stamp and returns the block closest to the requested time.
- Imagine how hideous this would be with a traditional, positional-parameter CALL interface! (Is it just me?)

```
IXGBRWSE REQUEST=READBLOCK,    Read block at lowest time stamp
         SEARCH=(R5),          ..Input time stamp
         GMT=NO,               ..Local time
         RETBLOCKID=EMRLGBLK,  ..Block ID return area
         BUFFER=(4),           ..Output buffer address
         BUFFLEN=EMRLGBFL,     ..Output buffer length
         BLKSIZE=SUBSWKF2,     ..Block size
         STREAMTOKEN=EMRLGSTK, ..Stream token
         BROWSETOKEN=EMRLGBTK, ..Browse token
         TIMESTAMP=SUBSWKD1,   ..Output time stamp
         MODE=SYNCECB,         ..Synchronous if possible
         ECB=SUBSWKF1,         ..ECB if not synchronous
         ANSAREA=SUBSLGAA,     ..Answer area address
         ANSLEN==A(Ansaa_Len), ..Answer area length
         MF=(E,SUBSW128)       ..Parameter list work area
```

Using a Macro to Map a Structure

- This is one of the two most recognized uses of a macro.
- SYS1.MACLIB and SYS1.MODGEN on a z/OS system are loaded with numerous examples of IBM-provided structure mappings: (CVT, IHAASCB, IKJTCB, IRARASD, etc.)
  - Warning! Modern IBM structure mappings are often created programmatically by a translator that generates bilingual mappings from PL/X source.
  - As such, they provide poor examples of how to define useful structure mappings in HLASM. The older mapping macros are better but still not exemplary because they are…older. ☺
- Mapping macros are better than COPY because you can use parameters to control if a DSECT is created, the prefix of the fields, print options, etc.

Using a Macro to Map a Structure

```
         PUSH  PRINT                                                    @D6A
         AIF   ('&LIST' EQ 'YES').ASCBLST                               @D6A
         PRINT OFF                                                      @D6A
.ASCBLST ANOP                                                           @D6A
         SPACE 1
         AIF ('&DSECT' EQ 'NO').ASCB10
ASCB     DSECT
         AGO .ASCB20
.ASCB10  ANOP
         DS    0D
ASCB     EQU   *
 .ASCB20 ANOP
ASCBEGIN DS    0D -                BEGINNING OF ASCB
ASCBASCB DS    CL4 -               ACRONYM IN EBCDIC -ASCB-
ASCBFWDP DS    A -                 ADDRESS OF NEXT ASCB ON ASCB READY
*                                  QUEUE
ASCBBWDP DS    A -                 ADDRESS OF PREVIOUS ASCB ON ASCB
*                                  READY QUEUE
ASCBLTCS DS    A -                 TCB and preemptable-class SRB   @07C
*                                  Local lock suspend service queue.
*                                  Serialization: ASCB CML promotion
*                                  WEB lock.
ASCBR010 DS    0D                  Reserved as of z/OS 1.12        @LLA
ASCBSUPC_PREZOS12 DS 0D -          SUPERVISOR CELL FIELD           @LLC
ASCBSVRB_PREZOS12 DS A  -          SVRB POOL ADDRESS.              @LLC
ASCBSYNC_PREZOS12 DS F -           COUNT USED TO SYNCHRONIZE SVRB POOL.
                                                                   @LLC
ASCBIOSP DS    A -                 POINTER TO IOS PURGE INTERFACE
*                                  CONTROL BLOCK (IPIB)
*                                  (MDC308)                  @Z40FP9A
```

Using a Macro To Add a New Instruction

- The MVC2 instruction works like MVC but uses the length of the source operand rather than of the target operand.

```
         MACRO ,
&LABEL   MVC2  &TARGET,&SOURCE
         PUSH  PRINT,NOPRINT
         PRINT OFF,NOPRINT
         MVC   &TARGET,&SOURCE
         ORG   *-6
         POP   PRINT,NOPRINT
&MVC2LEN SETA  L'&SOURCE                .Get source length
&I1      SETA  INDEX('&TARGET','(')     .Look for paren
         AIF   (&I1 GT 0).PAREN         .Branch if paren
&LABEL   MVC   &TARGET.(&MVC2LEN),&SOURCE
         MEXIT ,                        .Exit
.PAREN   ANOP  ,
&C1      SETC  '&TARGET'(1,&I1)         .Get left side
&C2      SETC  '&TARGET'(&I1+1,*)       .Get right side
&LABEL   MVC   &C1.&MVC2LEN.&C2,&SOURCE
         MEXIT ,                        .Exit
         MEND
```

Overriding Existing Instructions and Defining In-line Macros Within a Macro

- For one of our larger products:
  - We had a need to place program size into a field within the program's self-descriptive prefix.
  - When we restructured to use only relative branch, we needed to remove interspersed LTORGs and replace them with a single LTORG in the 'data' LOCTR.
- Changing all of these programs would have been a time consuming, menial, and error-prone task.
- We might have been able to write a program to read in the existing source code and write out changed source code.
- Instead we opted to use an already-existing common macro invocation at the top of the programs to define other macros to achieve these objectives.
- We have since leveraged this useful infrastructure for other things.

Overriding Existing Instructions and Defining In-line Macros Within a Macro

```
.DEFINE  ANOP  ,
&C1      SETC  '&LABEL'
         POP   PRINT,NOPRINT
         PRINT &EJESDAT
         EJESPEQU ,                        Define Phoenix derived equates
         PRINT &EJESSRC
         PUSH  PRINT,NOPRINT
         PRINT MCALL,NOPRINT
.*****************************************************************
EJESEND  OPSYN END                         Override END instruction     *
         MACRO ,                           Define END macro             *
&LABEL1  END   &LABEL2                     Macro prototype              *
         GBLB  &GBLTORG                    Global LTORG flag            *
         GBLC  &DATASEC                    Primary data section name    *
EOMMARK  LOCTR ,                           Create location counter
         DC    0D                          Align to doubleword
EOMPSIZE EQU   *-&DATASEC                  Calculate total program size
&DATASEC LOCTR ,                           Define primary section
         ORG   PROGSIZE                    Position to pgmsize field
         DC    A(EOMPSIZE)                 Define total program size
         ORG   ,
         AIF   (NOT &GBLTORG).EJESEN1      If not global LTORG          *

*****************************************************************
*                                                               *
*                    Global Literal Pool                        *
*                                                               *
*****************************************************************
&SYSECT  LOCTR ,                           Set LOCTR for constants
         EJESLTRG ,                        Define literal pool
.EJESEN1 ANOP  ,                           EndIf not global LTORG       *
         AIF   ('&LABEL2' EQ '').EJESEN2 If operand not null           *
&LABEL1  EJESEND &LABEL2                     Define end of program
         MEXIT ,                           Exit the macro               *
.EJESEN2 ANOP  ,                           Else operand is null         *
&LABEL1  EJESEND ,                           Define end of program
         MEXIT ,                           Exit the macro               *
.*                                         EndIf operand not null       *
         MEND  ,                           End of macro                 *
.*****************************************************************
         AIF   ('&LTORG' NE 'GLOBAL').DEFINE1
&GBLTORG SETB  1                           Show global LTORG
.*****************************************************************
EJESLTRG OPSYN LTORG                       Override LTORG instruction   *
         MACRO ,                           Define LTORG macro           *
&LABEL   LTORG ,                           Macro prototype              *
         MEND  ,                           End of macro                 *
.*****************************************************************
.DEFINE1 ANOP  ,
```

Using Macros To Define Tables

- Defining tables in assembler language is a time-consuming and error prone process requiring considerable manual effort; changes to the table structure can take a very long time to implement and exhaustively test.
- We use macros to define our tables. The parameters are provided as blank-delimited values that are AREAD by the macro, parsed as appropriate, and emitted as DCs.
- Optional: We also generate the table mapping DSECT from within the same macro. That way the code that generates the table and the code that generates the mapping are centralized.

SHARE
Technology · Connections · Results

Defining Tables The Old Way

- Manually-coded DCs are used to define the tables.
- This is a time-consuming and error-prone process.

```
*************************************************************
*                                                           *
*                Input Column Processing Table              *
*                                                           *
*************************************************************
          DS    0F
DJIFLD    DS    0CL16
          DC    AL1(EFLTDJJP,EJSVMJPR),AL2(EJSVAUTH-EJSDSECT)
          DC     A(DJJPX-*+4),A(0),AL2(EJHWJP),AL2(0)
          DC    AL1(EFLTDJJC,EJSVMJCL),AL2(EJSVAUTH-EJSDSECT)
          DC     A(DJJCX-*+4),A(0),AL2(EJHWJC),AL2(0)
          DC    AL1(EFLTDJSC,EJSVMSRV),AL2(EJSVAUTH-EJSDSECT)
          DC     A(DJSRVCX-*+4),A(0),AL2(EJHWSRVC),AL2(0)
DJIFLD#   EQU   (*-DJIFLD)/L'DJIFLD
          DC    F'-1'
```

SHARE
Technology · Connections · Results

Defining Tables The New Way (Via Macro)

```
*****************************************************************
*                                                               *
*                  Input Column Processing Table                *
*                                                               *
*****************************************************************
 DJ        EIFGEN START
-*Name   Len Id    Auth1   Auth2  Case Help  MTbl
- JP     8   DJJP  V MJPR  - -     Up  JP    No   JPRTY
- JC     8   DJJC  V MJCL  - -     Up  JC    No   JCLASS
- SRVC   8   DJSC  V MSRV  - -     Up  SRVC  No   SRVCLASS
-          EIFGEN END
+          DC    0F
+DJIFLD    DC    0CL16                Input column processing table
+          DC    AL1(EFLTDJJP)
+          DC      AL1(EJSVMJPR)
+          DC      AL2(EJSVAUTH-EJSDSECT)
+          DC      A(DJJPX-*+4)
+          DC      XL4'00'
+          DC      AL2(EJHWJP),AL2(0)
+          DC    AL1(EFLTDJJC)
+          DC      AL1(EJSVMJCL)
+          DC      AL2(EJSVAUTH-EJSDSECT)
+          DC      A(DJJCX-*+4)
+          DC      XL4'00'
+          DC      AL2(EJHWJC),AL2(0)
+          DC    AL1(EFLTDJSC)
+          DC      AL1(EJSVMSRV)
+          DC      AL2(EJSVAUTH-EJSDSECT)
+          DC      A(DJSRVCX-*+4)
+          DC      XL4'00'
+          DC      AL2(EJHWSRVC),AL2(0)
+DJIFLD#   EQU   (*-DJIFLD)/L'DJIFLD
+          DC    XL4'FFFFFFFF'
```

Writing A Translator/Compiler

- We have a macro that reads in source code statements using AREAD, translates that source code into assembler language statements, and then uses AINSERT to generate an equivalent assembler language routine as well as data elements in various LOCTRs that are used to support the execution of this special ‘language’.
- I really can’t go into detail about the nature of this code, what it looks like, or why we went to so much trouble.
- Suffice to say, this was no simple undertaking. The results have been spectacular!
- It simply would not have been possible without the remarkable power of the HLASM macro facility.

Using Structured Programming Macros—Such As Those From the IBM HLASM Toolkit

- This is a sample listing fragment showing SPM use. The "flow" bars are produced by my FLOWASM HLASM exit.

```
.                                          58489 ************************************************************
.                                          58490 * Search for Matching Column Name                          *
.                                          58491 ************************************************************
.0000325C 9200 83FC      000003FC          58492      MVI   SUBSWKH3,X'00'          Zero field TID value
.                                          58493      DO ,                          Do for column name search
.00003260 48E0 83F8               000003F8 58503      | LH    R14,SUBSWKH1          Get normalized length
.00003264 12EE                             58504      | DOEXIT LTR,R14,R14,NP       Exit if invalid length
.0000326A A7EE 0008               00000008 58517      | DOEXIT CHI,R14,GT,L'SUBSWKD1 Exit if too long
.00003272 D207 81C8 C4E8 000001C8 00003530 58530      | MVC   SUBSWKD1,=CL8' '      Blank out work field
.00003278 A7EA FFFF               FFFFFFFF 58531      | AHI   R14,-1                Make relative to zero
.0000327C 44E0 C4DA               00003522 58532      | EX    R14,MCLCOMV2          Copy to SUBSWKD1
.00003280 43E0 6000               00000000 58533      | IC    R14,EFLLSTID          Get list identifier
.00003284 A7EE 00C0               000000C0 58534      | IF CHI,R14,LT,EFLLSTIB      If tabular utility
.0000328C 06E0                             58548      | : BCTR  R14,0               Make relative to zero
.0000328E 5810 C4F0               00003538 58549      | : L     R1,=A(JJTUFLDIDX)   Point to index table
.00003292 A7F4 000E               000032AE 58550      | ELSE ,                      Else
.00003296 A7EA FF40               FFFFFF40 58558      | : AHI   R14,-EFLLSTIB       Make relative to base
.0000329A 95F2 A00B      0000000B         58559      | : IF CLI,EMRJES,EQ,EMRJES2  If running JES2
.000032A2 5810 C4F4               0000353C 58573      | : | L     R1,=A(J2TDFLDIDX)   Point to index table
.000032A6 A7F4 0004               000032AE 58574      | : ELSE ,                    Else running JES3
.000032AA 5810 C4F8               00003540 58582      | : | L     R1,=A(J3TDFLDIDX)   Point to index table
.                                          58583      | : ENDIF ,                   EndIf
.                                          58590      | ENDIF ,                     EndIf tabular utility
.000032AE 89E0 0003               00000003 58597      | SLL   R14,3                 Point to proper entry
.000032B2 1EE1                             58598      | LA    R14,0(R1,R14)         (same)
.000032B4 98EF E000               00000000 58599      | LM    R14,R15,0(R14)        Get offset & entry count
.000032B8 1EE1                             58600      | LA    R14,0(R14,R1)         Change offset into pointer
.                                          58601      | DO FROM=(R15)               Do for all entries
.000032BA D507 81C8 E000 000001C8 00000000 58614      | : DOEXIT CLC,SUBSWKD1,EQ,0(R14) Exit if matching entry
.000032C4 A7EA 0009               00000009 58627      | : LA    R14,FLD_TblLen(,R14)  Advance pointer
.000032C8 A7F6 FFF9               000032BA 58628      | ENDDO ,                     EndDo for all entries
.000032CC 12FF                             58638      | DOEXIT LTR,R15,R15,Z        Exit if column not found
.000032D2 D200 83FC E008 000003FC 00000008 58651      | MVC   SUBSWKH3(1),8(R14)    Copy field TID value
.                                          58652      ENDDO ,                       EndDo for column name search
```