How to Make Assembler Programs Easier to Read and Maintain Using Structured Programming Macros

Not Your Father’s Assembler Language

Edward E. Jaffe
Phoenix Software International

03 August 2010
Session 7175
Overview

• Structured programming overview.
• IBM’s HLASM structured programming macros.
• Tools I provide to assist HLASM programmers in writing structured programs.
• Code fragments and techniques.
• Conversion of existing code from unstructured to structured programming.
Structured Programming Overview
Structured Programming Disciplines

• Top-down development and design:
  • Program flow is always hierarchical.
  • Levels of abstraction become major routines or modules.
  • A routine or module must always return to its caller (which could be itself if recursive).
  • Major decision-making appears at as high a level as possible. The routine at the top of the hierarchy is a synopsis of the entire program.

• Programming in which few or no GOTOs are used:
  • Various combinations of only three basic programming structures – mathematically proven to solve any logic problem[1] – are used: sequence, choice, and repetition.
  • IBM’s Structured Programming Macros provide help in this area.

Evolution of GOTO Use in Programming Languages

• Today’s programming languages either discourage or completely disallow the use of GOTO statements. Those more recently invented are more likely to prohibit its use:
  • Fortran (1957) GOTO is required
  • Basic (1960) GOTO is required
  • C (1973) GOTO is used occasionally
  • Rexx (1981) GOTO is rarely or never used (not documented)
  • Ada (1983) GOTO is rarely or never used
  • C++ (1985) GOTO is rarely or never used
  • Perl (1987) GOTO is rarely or never used
  • Visual Basic (1991) GOTO is rarely or never used
  • Python (1991) has no GOTO statement
  • Java (1994) has no GOTO statement
  • Ruby on Rails (2004) has no GOTO statement
GOTO Density Metric

• The average number of lines of code between two GOTOs.
• Studies show that when sufficiently powerful programming structures are available, GOTOs are not used.
• My research into large assembler language programs showed just under 8 lines per GOTO (branch) excluding call/return.

<table>
<thead>
<tr>
<th>Files without GOTO</th>
<th>Fortran</th>
<th>C</th>
<th>Ada</th>
<th>PL8</th>
<th>HLASM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines/GOTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>none</th>
<th>81.5%</th>
<th>99.4%</th>
<th>98.5%</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>About 10 [3]</td>
<td>386</td>
<td>13614</td>
<td>1310</td>
<td>&lt;8</td>
<td></td>
</tr>
</tbody>
</table>

[2] PL8 is the language in which much IBM System z firmware is written.
[3] 8% - 13% of all Fortran statements are GOTOs.
Unstructured Programs: Customized Program Flow Can Become Complex “Spaghetti” Code
Structured Programs: Hierarchical Flow is Easier to Understand and Maintain

Any group of blocks in this diagram could be a single statement, construct, routine, subroutine, or a subprogram or module.
Nesting. The Most Important Element of Overall Structured Program Organization

- Nest!
  - Subroutines should not be created only to avoid code duplication. They should be the norm.

- Nest!
  - Implement a low-overhead save area stacking mechanism.

- Nest!
  - All routines should kept to a manageable size – no more than a couple/few of “pages” of code if possible.

- Don’t overdo it!
  - Like everything else in life, there are trade-offs. Gratuitous nesting can affect performance. Choose subroutine boundaries wisely, especially in performance sensitive code.
Flat Program Organization: Tedious to Follow; Every Branch Must Be Inspected

Task1    DC    0H
* Find the Entry
Find     DC    0H
        L     ...
        L     ...
Find1    DC    0H
        LTR   ...
        JNZ   Find2
        LHI   ...
        J     SetMsg2
        C     ...
        JE   Process
        L     ...
        J     Find1
* Process the Entry
Process   DC    0H
        MVC   ...
        L     ...
        ST     ...
        TM     ...
        JZ     Process1
        MVI   ...
        J     Account
        DC    0H
        MVC   ...
        MVC   ...
        CLC   ...
        JE     Account
        ST     ...
* Perform Accounting
Account   DC    0H
        L     ...
        L     ...
        LHI   ...
        J     SetMsg2
        Account1  DC    0H
        CLI   ...
        JE     SetMsg1
        L     ...
        A     ...
        ST     ...
        TM     ...
        JZ     Account2
        L     ...
        A     ...
        ST     ...
        Account2  DC    0H
        LTR   ...
        JH     SetMsg1
        LH     SetMsg1
        JCT   xx,Account1
        J     Task2
* Set Messages
SetMsg1   DC    0H
        ...
SetMsg2   DC    0H
        ...
Task2    DC    0H
Hierarchical Program Organization: Easier to Understand and Maintain

(mainline)

JAS R14,Task1
JAS R14,Task2
.
.
BR R14

*Perform Task 1*

Task1 DC 0H
STKSAVE PUSH
JAS R14,Task1Find
LTR R15,R15
JNZ Task1Msg
JAS R14,Task1Proc
LTR R15,R15
JNZ Task1Msg
JAS R14,Task1Acct
J Task1Ret

Task1Msg DC 0H
.
.

Task1Ret DC 0H
STKSAVE POP
BR R14

*Perform Task 2*

Task2 DC 0H
STKSAVE PUSH
.
.
STKSAVE POP
BR R14

Hierarchical Program Organization:
Easier to Understand and Maintain
IBM’s HLASM Structured Programming Macros
IBM’s Structured Programming Macros

• Delivered with the HLASM Toolkit—a licensed feature.
• Found in hlq.SASMMAC2 on z/OS systems.
• Activated simply by adding the above to SYSLIB and the following COPY statement to the top of your program:

  COPY ASMMSP Activate structured programming support

• Add the following if your program uses relative branching.

  SYSSTATE ARCHLVL=1 Program supports immediate/relative
   -OR-
  SYSSTATE ARCHLVL=2 Program supports z/Architecture
  ASMMREL ON Enable relative branch for SPMs
IBM’s Structured Programming Macros

• Leverage powerful HLASM capabilities.
  • HLASM macro support is extremely powerful. Most HLLs – even those that claim to support so-called “macros” – have no equivalent.
• Enforce program structure.
• Eliminate GOTO statements from program source.
• Eliminate extraneous labels.
• Eliminate out-of-line logic paths.
• Enhance source code readability.
• Enhance source code maintainability.
• Provide uniformity and standardization.
SPMs Enforce Program Structure

- SPMs define the set of building blocks (constructs) used to author the program.
- They provide enforcement necessary to prevent corruption of program structure.
- Requires no more programmer cooperation than do HLLs that support GOTO but discourage its use (e.g., Perl).
SPMs Eliminate GOTO Statements from Program Source

- As predicted by the studies, SPM use reduces the need/desire to code GOTO (BC and BRC instructions).
- Conditional branching is performed in accordance with the universally-understood rules of the construct. Control always returns back to the original path.
- **SPMs “hide” the branches that form the constructs.**
SPMs Eliminate Extraneous Labels

• Labels (other than those used for subroutines, labeled USINGs, etc.) represent unstructured exposures. The more labels that exist, the higher the probability that one or more of them will be used as the target of a branch.
• Label management (naming/renaming) is “busy work” and a constant source of programming errors.
• Code fragments copied from one part of a program to another require label “fix up”. Mistakes here can produce loops or worse.
• SPMs “hide” the labels that form the constructs.
SPMs Eliminate Out-of-line Logic Paths

- Out of line logic paths make programs harder to follow.
- *Every* conditional branch presents an opportunity to create out-of-line logic.
- I’ve seen situations where seemingly “unrelated” code was where I didn’t expect to see it.
- The longer the program, and the more labels it has, the easier it is to lose your perspective.
SPMs Enhance Source Code Readability

• SPMs facilitate code indentation – arguably the single most powerful heuristic ever devised for illustrating conditional program flow within source code.

• Editors (like that supplied with ISPF) are specifically designed to work with indented code such as that typically found in PL/I, C, C++, Pascal, Ada, Visual Basic, REXX, Perl, Java, etc.

• ISPF Editor features include:
  • Block change of indentation level.
  • Ability to exclude blocks of code from view.
  • Indentation level sensitive un-exclusion of lines in a block.
SPMs Enhance Source Code Maintainability

- When adding or removing cases, the code on the left has higher potential for introduction of “dumb” errors.

```
CLI 0(R1),value1
JNE LABELA
.
  . (code for value1)
  .
  J  LABELX
LABELA DC 0H
CLI 0(R1),value2
JNE LABELB
.
  . (code for value2)
  .
  J  LABELX
LABELB DC 0H
CLI 0(R1),value3
JNE LABELC
.
  . (code for value3)
  .
  J  LABELX
LABELC DC 0H
.
  . (handle all other cases)
  .
LABELX DC 0H
```
SPMs Provide Uniformity and Standardization

• SPMs reduce the number of different kinds of constructs used to write the program. They form the building blocks from which the program logic is constructed.
• No “custom” programming constructs are possible.
• Every programmer that reads or modifies the program understands *a priori* the flow of each construct without tedious inspection of the logic.
• Good programmers visualize their programs before they write them. Good programmers that use SPMs will visualize *structured* programs before they write them.
• Programmers learn to solve problems with the tools they are given. Programmers will actually *think* differently!
## SPM Mnemonics and Complements

<table>
<thead>
<tr>
<th>Case</th>
<th>Condition Mnemonics</th>
<th>Meaning</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>After compare instructions</td>
<td>H, GT, L, LT, E, EQ</td>
<td>High, Greater than, Low, Less than, Equal</td>
<td>NH, LE, NL, GE, NE</td>
</tr>
<tr>
<td>After arithmetic instructions</td>
<td>P, M, Z, O</td>
<td>Plus, Minus, Zero, Overflow</td>
<td>NP, NM, NZ, NO</td>
</tr>
<tr>
<td>After test under mask</td>
<td>O, M, Z</td>
<td>Ones, Mixed, Zeros</td>
<td>NO, NM, NZ</td>
</tr>
</tbody>
</table>
IF Macro Set: IF, ELSE, ELSEIF, ENDIF

IF

IF ELSE

IF ELSEIF
**IF Macro Set: IF, ELSE, ELSEIF, ENDIF**

IF ELSEIF ELSE

- If \( p \)
  - True: Logic
  - False: If \( q \)
    - True: Logic
    - False: Logic

**Note:** ELSEIF may be specified any number of times; ELSE only once.

**Predicate Values**

<table>
<thead>
<tr>
<th>Numeric value (1-14)</th>
<th>Condition mnemonic</th>
<th>Instruction,p1,p2,condition</th>
<th>Compare-instruction,p1,condition,p2</th>
</tr>
</thead>
</table>

**Connectors**

- AND
- OR
- ANDIF
- ORIF
IF – Basic Tests

IF CLI, 0(R1), GT, C' ' 
    ST R1, NBPtr 
ENDIF ,

+ CLI 0(R1), C' ' 
+ BRC 15-2,#@LB1 
ST R1, NBPtr 
+#@LB1 DC 0H

IF CLI, 0(R2), GT, C' ' 
    ST R2, NBPtr 
ELSE , 
    ST R2, BPtr 
ENDIF ,

+ CLI 0(R2), C' ' 
+ BRC 15-2,#@LB3 
ST R2, NBPtr 
+ BRC 15, #@LB5 
+#@LB3 DC 0H 
+#@LB5 DC 0H 
ST R2, BPtr
IF CLI,0(R1),GE,C'0',AND, CLI,0(R1),LE,C'9' OI Flag,Numeric ENDIF ,

+ CLI 0(R1),C'0'
+ BRC 15-11,#@LB6
+ CLI 0(R1),C'9'
+ BRC 15-13,#@LB6
OI Flag,Numeric
+#@LB6 DC 0H

IF CLI,0(R1),LT,C'0',OR, CLI,0(R1),GT,C'9'
NI Flag,X'FF'–Numeric
ENDIF ,

+ CLI 0(R1),C'0'
+ BRC 4,#@LB9
+ CLI 0(R1),C'9'
+ BRC 15-2,#@LB8
+#@LB9 DC 0H
NI Flag,X'FF'–Numeric
+#@LB8 DC 0H
IF – Logical Grouping With ANDIF

IF (CLI,0(R1),GT,C' '), OR,
   (LTR,R4,R4,NZ), AND,
   (CLC,SpecChar(2),EQ,0(R4)),
   ANDIF,
   (TM,Flag,FlagBit,NZ), AND,
   (CLM,R15,B'0011',LT,Limit), OR,
   (ICM,R2,B'1111',Offset,Z)
OI Flag,Passed
ENDIF,
IF – Logical Grouping With ORIF

IF (CLI,0(R1),GT,C' '),OR,
  (LTR,R4,R4,NZ),AND,
  (CLC,SpecChar(2),EQ,0(R4)),
ORIF,
  (TM,Flag,FlagBit,NZ),AND,
  (CLM,R15,B'0011',LT,Limit),OR,
  (ICM,R2,B'1111',Offset,Z)
OI Flag,Passed
ENDIF ,

+ CLI 0(R1),C' '
+ BRC 2,#@LB14
+ BRC R4,R4
+ BRC 15-7,#@LB13
+ CLC SpecChar(2),0(R4)
+ BRC 8,#@LB14
+ BRC 15-7,#@LB15
+ CLM R15,B'0011',Limit
+ BRC 4,#@LB14
+ ICM R2,B'1111',Offset
+ BRC 15-8,#@LB15
+#@LB14 DC 0H
+ TM Flag,FlagBit
+ BRC 15-7,#@LB15
+ CLM R15,B'0011',Limit
+ BRC 4,#@LB14
+ ICM R2,B'1111',Offset
+ BRC 15-8,#@LB15
+#@LB14 DC 0H
+TM Flag,Passed
+#@LB15 DC 0H
IF – Nesting With ELSEIF

```assembly
IF CLI,0(R1),EQ,C'0'
    LA    R15,12
ELSE ,
    IF CR,R2,EQ,R3
        LA    R15,16
    ELSE ,
        IF CLC,=Y(Big),GT,Size
            LA    R15,24
        ELSE ,
            XR    R15,R15
        ENDIF ,
    ENDIF ,
ELSE ,
    XR    R15,R15
ENDIF ,
```
DO Macro Set: DO, DOEXIT, ASMDELETE, ITERATE, ENDDO

DO ,

DO INF

DO UNTIL or FROM, TO, BY

DO WHILE
## DO – Loop Terminator Generation

<table>
<thead>
<tr>
<th>Type</th>
<th>Keywords</th>
<th>Other Conditions</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>None</td>
<td>ONCE parameter or no parameters (null comma)</td>
<td>No terminator</td>
</tr>
<tr>
<td>Infinite loop</td>
<td>Neither FROM, WHILE, nor UNTIL</td>
<td>INF parameter</td>
<td>BC 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BRC 15</td>
</tr>
<tr>
<td>Explicit Specification</td>
<td>FROM, plus TO and/or BY</td>
<td>BXH/BRXH parameter</td>
<td>BXH, BRXH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BXLE/BRXLE parameter</td>
<td>BXLE, BRXLE</td>
</tr>
<tr>
<td>Counting</td>
<td>FROM only</td>
<td>Two or three values</td>
<td>BCT, BCTR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BRCT, BRCTR</td>
</tr>
<tr>
<td>Backward Indexing</td>
<td>FROM, TO and BY</td>
<td>FROM and TO numeric, FROM value &gt; TO value</td>
<td>BXH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BRXH</td>
</tr>
<tr>
<td>Backward Indexing</td>
<td>FROM BY</td>
<td>BY numeric and less than zero</td>
<td>BXH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BRXH</td>
</tr>
<tr>
<td>Forward Indexing</td>
<td>All other combinations</td>
<td></td>
<td>BXLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BRXLE</td>
</tr>
</tbody>
</table>
## DO – Register Initialization

<table>
<thead>
<tr>
<th>Value Given</th>
<th>Instruction Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None (passed in)</td>
</tr>
<tr>
<td>Zero</td>
<td>SR Rx,Rx</td>
</tr>
<tr>
<td>0 to 4095</td>
<td>LA Rx,value</td>
</tr>
<tr>
<td>-32768 to −1 or 4096 to 32767</td>
<td>LHI Rx,value or LH Rx,=H'value'</td>
</tr>
<tr>
<td>Other numbers</td>
<td>L Rx,=F'value'</td>
</tr>
<tr>
<td>(value)</td>
<td>LR Rx,value</td>
</tr>
<tr>
<td>Other</td>
<td>L Rx,Other</td>
</tr>
</tbody>
</table>
DO – Basic Formats

Simple

DO , JAS  R14,ProcessInput ENDDO , +#@LB21 DC  0H JAS  R14,ProcessInput

Infinite

DO INF JAS  R14,ProcessTillDead ENDDO , +#@LB18 DC  0H JAS  R14,ProcessTillDead + BRC 15,#@LB18
DO – Backward Index (Implied BXH)

DO FROM=(R1,100), TO=(R5,1),
    BY=(R4,-1)
    STC   R1,0 (R1,R2)
ENDDO ,

DO FROM=(R1,100), BY=(R5,-1)
    STC   R1,0 (R1,R2)
ENDDO ,

+       LA    R1,100
+       LA    R5,1
+       LHI   R4,-1
+       @LB38 DC    0H
+       @LB39 DC    0H
+       BRXH  R1,R4,#@LB38

+       LA    R1,100
+       LHI   R5,-1
+       @LB41 DC    0H
+       @LB42 DC    0H
+       BRXH  R1,R5,#@LB41
DO – Forward Index (Implied BXLE)

DO FROM=(R1,1), TO=(R5,100),
  BY=(R4,1)
  STC R1,0 (R1,R2)
ENDDO,

DO FROM=(R1,ArrayFirst),
  TO=(R5,ArrayLast),
  BY=(R4,=A(EntryLen))
  JAS R14,ProcessEntry
ENDDO,

+  LA R1,1
+  LA R5,100
+  LA R4,1
+#@LB47  DC 0H
  STC R1,0 (R1,R2)
+#@LB48  DC 0H
+  BRXLE R1,R4,#@LB47

+  L R1,ArrayFirst
+  L R5,ArrayLast
+  L R4,=A(EntryLen)
+#@LB44  DC 0H
  JAS R14,ProcessEntry
+#@LB45  DC 0H
+  BRXLE R1,R4,#@LB44
DO – Explicit BXH/BXLE

DO BXLE, FROM=(R1,1), TO=(R15,100),
    BY=(R14,1)
    STC  R1, 0 (R1,R2)
ENDDO ,

DO BXH, FROM=(R1,ArrayLast),
    TO=(R5,ArrayFirst),
    BY=(R4,=A(-EntryLen))
    JAS  R14, ProcessEntry
ENDDO ,
DO – Counting

LHI   R0,MaxItems
DO FROM=(R0)
   A   R14,0(,R1)
   LA  R1,4(,R1)
ENDDO ,

DO FROM=(R0,MaxItems)
   A   R14,0(,R1)
   LA  R1,4(,R1)
ENDDO ,
DO – While and Until

DO WHILE=(CLI,0(R1),LE,C' ')
   AHI R1,1
ENDDO ,

DO UNTIL=(CLI,0(R1),GT,C' ')
   AHI R1,1
ENDDO ,

+ BRC 15,#@LB50
+#@LB51 DC 0H
+#@LB50 DC 0H
+ CLIP 0(R1),C' ' + BRC 13,#@LB51

+#@LB55 DC 0H
+#@LB56 DC 0H
+ CLIP 0(R1),C' ' + BRC 15-2,#@LB55
DO – Combining Other Keywords With While and/or Until

DO FROM=(R0),
    WHILE=(CLI,0(R1),LE,C' ')
    AHI R1,1
ENDDO ,

DO WHILE=(CLI,0(R1),LE,C' ')
    UNTIL=(LTR,R15,R15,NZ)
    AHI R1,1
    JAS R14,ProcessChar
ENDDO ,
DO – Demand Exit

DOEXIT conditions[,DO=do_label]
ASMLEAVE [do_label]

OUTR DO UNTIL=(LTR,R15,R15,NZ)
   DO FROM=(R0)
      DOEXIT CLI,0(R1),GT,C' '
         JAS R14,ProcessChar
      IF LTR,R15,R15,NZ
         MVI FootPrint,C'C'
         ASMLEAVE OUTR
         ENDF
         AHI R1,1
      ENDDO
      JAS R14,ProcessKwd
      ENDDO
   +@LB77 DC 0H
   +@LB82 DC 0H
   + CLI 0(R1),C' '
   + BRC 2,#@LB81
   JAS R14,ProcessChar
   + LTR R15,R15
   + BRC 15-7,#@LB76
   MVI FootPrint,C'C'
   + BRC 15,#@LB76
   +@LB86 DC 0H
   AHI R1,1
   +@LB83 DC 0H
   + BRCT R0,#@LB82
   +@LB81 DC 0H
   JAS R14,ProcessKwd
   + LTR R15,R15
   + BRC 15-7,#@LB77
   +@LB76 DC 0H
DO – Demand Iteration

ITERATE [do_label]

OUTR DO INF

JAS R14,GetStmt
DOEXIT LTR,R15,R15,NZ
DO FROM=(R0)
  JAS R14,ProcessKwd
  IF LTR,R15,R15,NZ
    ITERATE OUTR
  ENDF
  AHI R1,1
ENDDO
JAS R14,PutResults
ENDDO,

+#@LB89 DC 0H
JAS R14,GetStmt
+
LTR R15,R15
+
BRC 7,#@LB88
+#@LB93 DC 0H
JAS R14,ProcessKwd
+
LTR R15,R15
+
BRC 15-7,#@LB95
+
BRC 15,#@LB89
+#@LB95 DC 0H
AHI R1,1
+#@LB94 DC 0H
+
BRCT R0,#@LB93
JAS R14,PutResults
+
BRC 15,#@LB89
+#@LB88 DC 0H
DO – Alternate Labeling Method

ProcessKwds DO ,
    JAS     R14,GetNextKwd
    .
    ASMDELETE ProcessKwds
    .
    ITERATE ProcessKwds
    .
    ENDDO ,

    Do for keyword processing
    Get next keyword
    .
    Finished with keywords
    .
    Process next keyword
    .
    EndDo for keyword processing

DO LABEL=ProcessKwds
    JAS     R14,GetNextKwd
    .
    ASMDELETE ProcessKwds
    .
    ITERATE ProcessKwds
    .
    ENDDO ,

    Do for keyword processing
    Get next keyword
    .
    Finished with keywords
    .
    Process next keyword
    .
    EndDo for keyword processing
CASE Macro Set: CASENTRY, CASE, ENDCASE

Notes:
• Values in register x are powers of 2 (i.e., 1s, 2s, 4s, 8, 16s, etc.).
• Control passed via branch table. Very efficient for processing many uniformly distributed numeric values.
• Value of zero not supported (unfortunately).
• R0 destroyed when relative branch used.
CASE – Relative Branch Version

CASENTRY R15
CASE 1
  JAS R14,HandleCase1
CASE 2
  JAS R14,HandleCase2
CASE 5
  JAS R14,HandleCase5
ENDCASE ,

Note: When SYSSTATE ARCHLVL=2 is not in effect, the blue fragment expands to:

+ LR 0,R15
+ CNOP 0,4
+ BRAS R15,++8
+ DC A(#@LB118-*)
+ AL R15,0(R15,0)
SELECT Macro Set: SELECT, WHEN, NEXTWHEN, OTHERWISE, ENDSERL

```
When 1
True

Logic 1

When 2
True

Logic 2

NEXTWHEN

Logic m

Otherwise
```

SHARE in Boston
SELECT – Global Test

SELECT CLI, 0(R1), EQ
WHEN C'A'
  LHI R15,12
WHEN C'B'
  LHI R15,16
WHEN C'C'
  LHI R15,24
WHEN C'D'
  LHI R15,8
OTHERWISE ,
  XR R15, R15
ENDSEL ,

+ CLI 0(R1), C'A'
+ BRC 15-8, @LB145
  LHI R15,12
+ BRC 15, @LB144
  DC 0H
+ @LB145
+ CLI 0(R1), C'B'
+ BRC 15-8, @LB147
  LHI R15,16
+ BRC 15, @LB144
  DC 0H
+ @LB147
+ CLI 0(R1), C'C'
+ BRC 15-8, @LB149
  LHI R15,24
+ BRC 15, @LB144
  DC 0H
+ @LB149
+ CLI 0(R1), C'D'
+ BRC 15-8, @LB151
  LHI R15,8
+ BRC 15, @LB144
  DC 0H
+ @LB151
+ XR R15, R15
+ @LB144 DC 0H
SELECT – Unique Tests

SELECT ,
WHEN CLI,0(R1),EQ,0
  LHI R15,12
WHEN CLI,0(R2),EQ,1
  LHI R15,16
WHEN CLI,0(R3),EQ,2
  LHI R15,24
WHEN CLI,0(R4),EQ,9
  LHI R15,8
 OTHERWISE ,
XR R15,R15
ENDSEL ,
+ CLI 0(R1),0
+ BRC 15-8,#@LB136
  LHI R15,12
+ BRC 15,#@LB135
+#@LB136 DC 0H
+ CLI 0(R2),1
+ BRC 15-8,#@LB138
  LHI R15,16
+ BRC 15,#@LB135
+#@LB138 DC 0H
+ CLI 0(R3),2
+ BRC 15-8,#@LB140
  LHI R15,24
+ BRC 15,#@LB135
+#@LB140 DC 0H
+ CLI 0(R4),9
+ BRC 15-8,#@LB142
  LHI R15,8
+ BRC 15,#@LB135
+#@LB142 DC 0H
XR R15,R15
+#@LB135 DC 0H
SEARCH Macro Set: STRTSRCH, EXITIF, ORELSE, ENDLOOP, ENDSRCH

Notes:
- STRTSRCH has same loop control options as DO.
- ENDLOOP (Logic D) differentiates SEARCH from DO.
- DOEXIT and ASMLEASE go to ENDLOOP logic.
- EXITIF and ORELSE are optional.
- Each EXITIF (except the last) must be followed by an ORELSE.
Why We Never Use SEARCH

• SEARCH has no direct counterpart in other structured programming languages. It is unique to HLASM SPMs. For this reason, I prefer that our programmers not use it.

• At one time SEARCH was necessary to address deficiencies in the DO macro set.
  • No simple DO.
  • No DOEXIT support for compound tests.
  • No DOEXIT/ASMLEAVE/ITERATE to outer (labeled) DOs from within inner DOs or other constructs.

• These and other similar deficiencies have since been resolved.
Code Fragments and Techniques
Simple DO Exclusion Tests

- This routine updates the record count if a record exists. ProcessDetail routine invoked only for records that are not headers or trailers.
- Logic identical to what would be coded in “traditional” assembler language. No additional overhead whatsoever.

```
DO ,                          Do for record
   ICM R3,B'1111',RecPtr      Get record address
   DOEXIT Z                   Exit if no record
   L    R0,RecCount          Get record count
   AHI  R0,1                  Add 1
   ST   R0,RecCount          Update record count
   DOEXIT CLI,RecType,EQ,RecHdr Exit if header
   DOEXIT CLI,RecType,EQ,RecTrl Exit if trailer
   JAS    R14,ProcessDetail  Process detail record
ENDDO ,                       EndDo for record
```
Simple DO Mainline

- Below is an example of a routine that calls many subroutines. Checking return codes is exclusion.
- Again, exactly the same logic as “traditional” code, but without the ever-present temptation to branch “wildly”.

```plaintext
DO LABEL=MainLine
   JAS   R14,FindIt         Locate the instance
   DOEXIT LTR,R15,R15,NZ    Exit if error
   JAS   R14,Modify         Modify the instance
   DOEXIT LTR,R15,R15,NZ    Exit if error
   JAS   R14,AcctUpdt       Update accounting info
   DOEXIT LTR,R15,R15,NZ    Exit if error
   JAS   R14,Unlock         Unlock the data base
   DOEXIT LTR,R15,R15,NZ    Exit if error
   JAS   R14,Report         Generate report data
   DOEXIT LTR,R15,R15,NZ    Exit if error

  . (Insert additional calls here)

ENDDO , MainLine
```

EndDo for mainline
Nested Simple DO Inclusion Tests

- Inner DO contains tests that include, rather than exclude.
- In this case, reasons why a message *should* be formatted.

```plaintext
DO LABEL=SetVarsMsg              Do for msg processing
  DO ,                          Do for msg include tests
    DOEXIT CLI,CurMsgType,LE,C' ' Include if no msg yet formatted
    DOEXIT TM,MsgFlgs,Error,O     Include if an error message
      . (other include tests)    .
      . ASMLEAVE SetVarsMsg       Bypass message formatting
      ENDDO ,                     EndDo for msg include tests
      . (logic to format the message)
      .
  ENDDO , SetVarsMsg              EndDo for msg processing
```
Iterative Processing Loop

XR    R4,R4                  Zero entries read
XR    R5,R5                  Zero entries processed
DO INF                       Do for all entries
    JAS   R14,GetEntry       Get the next entry
    DOEXIT LTR,R15,R15,NZ    Exit if no more entries
AHI   R4,1                   Increment entries read
SELECT CLI,EntryType,EQ     Select entry type
    WHEN EntryTypeA
        JAS   R14,ProcessTypeA  Process TypeA entry
    WHEN EntryTypeB
        JAS   R14,ProcessTypeB  Process TypeB entry
    (other WHEN clauses as needed)
    OTHERWISE ,  ITERATE ,   Otherwise unrecognized
                ENDSEL ,       Skip unrecognized entries
        AHI   R5,1,              EndSel entry type
    ENDDO ,                     Increment entries processed
                                            EndDo for all entries
Combining SPM Condition Tests With Instructions That Set the CC

- This fragment copies the job name associated with an ASCB on a z/OS system into the field called ESMFJOBN.

```
L R14,GENASCB
USING ASCB,R14 *** Synchronize ASCB
IF LT,R15,ASCBJBNI,NZ If job name available
  MVC ESMFJOBN,0(R15) Set job name
ELSE ,
  IF LT,R15,ASCBJBNS,NZ If task name available
    MVC ESMFJOBN,0(R15) Set as job name
  ELSE ,
    MVC ESMFJOBN,=C'*UNKNOWN' Set name to '*UNKNOWN'
  ENDIF ,
ENDIF ,
DROP R14 *** Drop ASCB
```
Combining SPM Condition Tests With Macros That Set the CC

- The SPMs don’t know the full instruction set. (They recognize only a subset of instructions for special handling.) You can take advantage of this fact.[1]

```asm
MACRO ,, $NSXENCL ,
$NSXCALL PCVTSSEOT,PARMS=SET   Invoke enclave eligibility
LTR   R15,R15                   Test return code
MEND  ,
.
.
IF $NSXENCL,0,0,NZ
   JAS   R14,BadEnclaveSet
ELSE ,
   (process logic in enclave)
ENDIF ,
```

[1] Thanks to Tom Harper for pointing this out.
It Seems That SELECT with Unique Tests And IF/ELSEIF Are Identical Constructs

- This example also illustrates that DOEXIT/ASMELAVE may be nested anywhere within inner non-DO structures.

```
DO,
  SELECT ,
  WHEN CHI,R1,EQ,0   Select Start value
    MVC EMRPARMS,=F'1'   Force to top of data
  WHEN CL,R1,EQ,X'7FFFFFFF' When Start=LAST (explicit)
    MVC EMRPARMS,=X'7FFFFFFF' Set both values to LAST
    MVC EMRPARMS+4,=X'7FFFFFFF' (same)
  ASMELAVE ,   All processing complete
  WHEN CHI,R1,EQ,-1 When Start=Current
    MVC EMRPARMS,CBLKATNM  Set to absolute number at top
  WHEN CHI,R1,EQ,-2 When Start=Time/Date (unsupported)
    MVC EMRPARMS,=F'1'  Force to top of data
  WHEN CHI,R1,LT,0 When Start=Label
    MVC EMRPARMS,0(R1)  Set value at label
  OTHERWISE ,
    AL R1,CBLKBNDL 
    AHI R1,-1  
    ST R1,EMRPARMS  
  ENDSel ,  EndSel Start value
  . (additional processing for all but one case)
  .
ENDDO ,  EndDo for Start value
```
It Seems That SELECT with Unique Tests And IF/ELSEIF Are Identical Constructs

• The choice of which to use seems to depend entirely on which you find more appropriate/readable.

DO,
  
  IF CHI,R1,EQ,0                           Do for Start value
    MVC EMRPARMS,=F'1'
    ELSEIF CL,R1,EQ,=X'7FFFFFFFF'
    MVC EMRPARMS,=X'7FFFFFFFF'
    MVC EMRPARMS+4,=X'7FFFFFFFF'          (same)
    ASMLEASE                            All processing complete
    ELSEIF CHI,R1,EQ,-1
    MVC EMRPARMS,CBLKTATNM
    ELSEIF CHI,R1,EQ,-2
    MVC EMRPARMS,=F'1'
    ELSEIF CHI,R1,LT,0
    MVC EMRPARMS,0(R1)
    ELSE                                    Else Start=ordinary numeric
    AL R1,CBLKBNDL                     Make relative to low boundary
    AHI R1,-1                           (same)
    ST R1,EMRPARMS                     (same)
  ENDDO,                                EndIf Start=FIRST

  . (additional processing for all but one case)

  ENDDO,                                EndDo for Start value
NEXTWHEN Provides SELECT With Additional Capability

- NEXTWHEN goes to the next clause (WHEN or OTHERWISE).
- NEXTWHEN may appear anywhere—even within nested constructs such as IF or DO.
- This capability does not exit in IF/ELSEIF. Depending on what's being done, duplicate logic could be required. 😞
Getting SPMs Inside Macros to Print

- Thankfully, the SPMs explicitly disable printing of their own inner macro calls using PRINT NOMCALL.
- Use PRINT MCALL to ensure SPM invocations appear on the assembler listing when PRINT GEN is used.

```assembly
MACRO 'TESTMAC
    PUSH PRINT,NOPRINT
    PRINT MCALL,NOPRINT
    XR   R15,R15
    IF CLI,0(R1),EQ,C'X'
        LHI   R15,4
    ENDIF ,
    POP   PRINT,NOPRINT
    MEXIT ,
    MEND ,

    TESTMAC ,
    XR   R15,R15
    IF CLI,0(R1),EQ,C'X'
        CLI       0(R1),C'X'
        BRC       15-8,#@LB1
        LHI   R15,4
    ENDIF ,
    #@LB1   DC   0H
```

Thankfully, the SPMs explicitly disable printing of their own inner macro calls using PRINT NOMCALL.

Use PRINT MCALL to ensure SPM invocations appear on the assembler listing when PRINT GEN is used.
Customizing the Macro Names

- Make modifications to hlq.SASMMAC2(ASMMNAME)

```plaintext
&ASMA_NAMES_CASE     SETC 'CASE'                                  00044000
&ASMA_NAMES_CASENTRY SETC 'CASENTRY'                             00045000
&ASMA_NAMES_DO       SETC 'DO'                                    00046000
&ASMA_NAMES_DOEXIT   SETC 'DOEXIT'                                00047000
&ASMA_NAMES_ELSE     SETC 'ELSE'                                  00048000
&ASMA_NAMES_ENDCASE  SETC 'ENDCASE'                               00049000
&ASMA_NAMES_ENDDO    SETC 'ENDDO'                                 00050000
&ASMA_NAMES_ENDIF    SETC 'ENDIF'                                 00051000
&ASMA_NAMES_ENDLOOP  SETC 'ENDLOOP'                               00052000
&ASMA_NAMES_ENDSEL   SETC 'ENDSEL'                                00053000
&ASMA_NAMES_ENDSRCH  SETC 'ENDSRCH'                               00054000
&ASMA_NAMES_EXITIF   SETC 'EXITIF'                                00055000
&ASMA_NAMES_IF       SETC 'IF'                                    00056000
&ASMA_NAMES_ORELSE   SETC 'ORELSE'                                00057000
&ASMA_NAMES_OTHRWISE SETC 'OTHRWISE'                              00058000
&ASMA_NAMES_SELECT   SETC 'SELECT'                                00059000
&ASMA_NAMES_STRTSRCH SETC 'STRTSRCH'                              00060000
&ASMA_NAMES_WHEN     SETC 'WHEN'                                  00061000
&ASMA_NAMES_ELSEIF   SETC 'ELSEIF'                                00062000
&ASMA_NAMES_LEAVE    SETC 'LEAVE'                                 00063000
&ASMA_NAMES_ITERATE  SETC 'ITERATE'                              00064000
&ASMA_NAMES_NEXTWHEN SETC 'NEXTWHEN'                              00065000
```
Our Structured Source Record Layout

- Long (but reasonable) labels used for major routines.
- Short labels (4 chars or less) for labeled USINGs.
- “Zero-indent” operation code begins in column 6, not 10.
- “Zero-indent” operand begins in column 12, not 16.
- “Zero-indent” commentary begins in standard column 36.
- Indentation delta is always 2 bytes.
- Comment blocks for subroutines start in column 1.
- Small comment blocks for code fragments follow indentation.
Our Structured Source Record Layout

Perform UNIT Modifications

ModifyUnit DC 0H
  STKSAVE PUSH                  Save the registers
  LARL  R12,ModifyUnitConst     Point to constants
  USING ModifyUnitConst,R12     Synchronize base register

Get Specified Value

MVI   LIFLDTID,EFLTLIUN       Set field text unit ID
  EJESSRV TYPE=GETBOVR,
      PARM=EFLTLIUN       Get batch overtype value
                  (same)
  XR    R15,R15                 Zero out message number
  IF CLI,LIUNIT,GT,C' '         If value supplied

Validate the Value

DO ,                          Do for validation
  IF CLI,LIUNIT,EQ,C'S'        If SNA requested
      MVC2  LIUNIT,=CL4'SNA'    Set to SNA
      ASMDELETE ,               Done with validation
      ENDF ,                     EndIf SNA requested
      . (more code follows ...)
Some Helpful Rules of Thumb

• Avoid the use of vectored returns.
  • Vectored returns imply a branch table follows the subroutine linkage. Branch tables imply GOTOs (branches) and labels.
• Try to make USING/DROP and PUSH/POP happen at the same indentation level.
• Use VECTOR=B for CASE macro set when using based branches. (Or just always use relative branches.)
• Choose constructs that require minimal changes to add new cases in the future.
  • Think about the next programmer – even if it’s you!
• Avoid excessive indentation.
Some Helpful Rules of Thumb

• Don’t be afraid to insert “white space” between statements.
• Use large screens when editing (I use 90x80).
  • The larger the screen, the more logic you can see at once.
• Keep the size of constructs “reasonable”.
  • Ideally, a construct will fit on one “page” so you can see the boundaries. A couple/few “pages” is not unreasonable.
  • A “page” of code is whatever size you decide it should be. We assemble with LINECOUNT(100).
  • Very large CASE or SELECT structures should have a comment block precede each CASE/WHEN clause. That clause can be about the size of any other “normal” routine.
• Create subroutines when things start to get unwieldy.
Make Trivial THEN Clause Rather Than ELSE (Just My Personal Preference)

• Instead of this:

  JAS  R14,MySubroutine        Call the subroutine
  IF LTR,R15,R15,Z             If subroutine failure
  .
  .  (lots of code – perhaps crossing “page” boundary)
  .
  ELSE ,                        Else subroutine failed
  MVC    FAILRSN,=CL8’MySub’    Set failure reason
  ENDIF ,                       EndIf all went well

• I prefer to see this because I see both paths immediately:

  JAS  R14,MySubroutine        Call the subroutine
  IF LTR,R15,R15,NZ             If subroutine failure
  MVC    FAILRSN,=CL8’MySub’    Set failure reason
  ELSE ,                        Else everything AOK
  .
  .  (lots of code – perhaps crossing “page” boundary)
  .
  ENDIF ,                       EndIf subroutine failure
Challenges Caused by Assembler Language Syntax Restrictions

- Existing assembler language syntax rules are not conducive to free-form indentation.
  - Continuation characters must appear in column 72.
  - Continued statements must begin in column 16.
  - Comment statements must have an asterisk (*) in column 1.
- Shifting a block of code left or right to change the indentation level often creates syntax errors.
- My FLOWASM HLASM exit addresses these issues.
- FLOWASM is written using SPMs but does not depend on itself for obvious reasons.
Tools I Provide to Help HLASM Programmers Write Structured Programs
Assembler Language Programming
Resources I’ve Made Public

• Minor modifications to the SPMs.
• STKSAVE Macro
  • Originally based on—but not actually the same as—a macro we use internally.
• FLOWASM HLASM Exit
  • *Exactly* the same exit we and some other ISVs use internally.

Available from:
ftp://ftp.phoenixsoftware.com/pub/demo/flowasm.xmi
ftp://ftp.phoenixsoftware.com/pub/demo/flowasm.zip
Modifications to the SPMs

- **EEJASM1**—for HLASM 1.4 users
  - APAR PK01283 assumed
  - NEXTWHEN macro
  - C/NC for carry/nocarry\(^1\) and B/NB for borrow/noborrow when testing condition codes after logical operations.

- **EEJASM2**—for HLASM 1.5 users
  - Same as EEJASM1 but reworked for HLASM 1.5.

- **EEJASM3**—for HLASM 1.6 users
  - C/NC for carry/nocarry\(^1\) and B/NB for borrow/noborrow when testing condition codes after logical operations.

\(^1\) The C/NC modification contributed by Tom Harper and used with his permission.
STKSAVE Macro

• Low-overhead *local* save area stack services.
• Can optionally save/restore access registers.
• Can save/restore any subset of registers.
• Requires 32-byte stack control area.
  • Initialized by INIT call at program startup.
• Currently for 24/31-bit mode only.
• Originally based on—but not actually the same as—a macro we use internally.
FLOWASM HLASM Exit

- Relaxes cumbersome syntax rules:
  - Comment blocks may start in any column; start with * or /*
  - No explicit continuation needed when macro operands ends with a trailing comma.
  - Continued macro operands may start in any column.
- For z/OS allows both fixed and variable source (SYSIN):
  - Variable length input may be numbered or unnumbered
  - Variable length explicit continuation is trailing + character.
  - Library (SYSLIB) still restricted to LRECL=80.
- Prints “flow bars” to match up SPMs on HLASM listing.
- Also works with HLASM on z/VM and z/VSE.
FLOWASM Reformatting Too-Long Lines

• Remove superfluous blanks between op-code and operand.
• If still too long, remove superfluous blanks between operand and commentary.
• If still too long, remove superfluous blanks before op-code.
• If still too long:
  • If operand fits on the line, commentary is truncated.
  • If operand is too long, it is wrapped and continued in column 16 of the next line along with the commentary.
FLOWASMS Automatic Continuation

- Detects trailing comma on macro operand and supplies - continuation character.
- Continued operand shifted into column 16.
  - If commentary must be moved, it is moved immediately after operand.
  - If line too long, reformat as described on previous slide.
58489  ************************************************************
58490  * Search for Matching Column Name                           *
58491  ************************************************************
58492  MVI  SUBSWKH3,X'00'  Zero field TID value
58493  DO , Do for column name search
58494  | LA    R14, SUBSWKH1  Get normalized length
58495  | DOEXIT LTR, R14, R14, NP Exit if invalid length
58496  | DOEXIT CHI, R14, GT, L'SUBSWKD1 Exit if too long
58497  | MVC    SUBSWKD1, =CL8' ' Blank out work field
58498  | AHI    R14, -1 Make relative to zero
58499  | EX     R14, MCLCOMV2 Copy to SUBSWKD1
58500  | IC      R14, EFLLISTID Get list identifier
58501  | IF CHI, R14, LT, EFLLLSTIB If tabular utility
58502  | : BCTR R14, 0 Make relative to zero
58503  | : L     R1, =A(JJTUFLDIDX) Point to index table
58504  | ELSE , Else running JES2
58505  | : AHI    R14, -EFLLLSTIB Make relative to base
58506  | : IF CLI, EMRJES, EQ, EMRJES2 If running JES2
58507  | : : L    R1, =A(J2TDFLDIDX) Point to index table
58508  | : ELSE , Else running JES3
58509  | : : L    R1, =A(J3TDFLDIDX) Point to index table
58510  | ENDIF , EndIf tabular utility
58511  | ENDIF , EndIf for all entries
58512  | DO FROM=(R15) Do for all entries
58513  | : DOEXIT CLC, SUBSWKD1, EQ, 0(R14) Exit if matching entry
58514  | : : LA    R14, FLD_TblLen(R14) Advance pointer
58515  | : ENDDO , EndDo for all entries
58516  | : MVC    SUBSWKH3(1), 8(R14) Copy field TID value
58517  | ENDDO , EndDo for column name search
58518
58530  MVC SUBSWKD1, =CL8' ' Blank out work field
58531  AHI R14, -1 Make relative to zero
58532  EX R14, MCLCOMV2 Copy to SUBSWKD1
58533  IC R14, EFLLISTID Get list identifier
58534  IF CHI, R14, LT, EFLLLSTIB If tabular utility
58535  : BCTR R14, 0 Make relative to zero
58536  : L R1, =A(JJTUFLDIDX) Point to index table
58537  ELSE , Else running JES2
58538  : AHI R14, -EFLLLSTIB Make relative to base
58539  : IF CLI, EMRJES, EQ, EMRJES2 If running JES2
58540  : : L R1, =A(J2TDFLDIDX) Point to index table
58541  : ELSE , Else running JES3
58542  : : L R1, =A(J3TDFLDIDX) Point to index table
58543  ENDIF , EndIf tabular utility
58544  ENDIF , EndIf for all entries
58545  DO FROM=(R15) Do for all entries
58546  : DOEXIT CLC, SUBSWKD1, EQ, 0(R14) Exit if matching entry
58547  : : LA R14, FLD_TblLen(R14) Advance pointer
58548  : ENDDO , EndDo for all entries
58549  : MVC SUBSWKH3(1), 8(R14) Copy field TID value
58550  ELSE , Else for column name search
58551  ENDDO , EndDo for column name search
Conversion of Existing Programs From Unstructured to Structured Programming
Structured Code Represents a Subset of All Possible Programming Constructs
Conversion Difficulty is a Function of How Many Custom Constructs Are Used

- Converting structured code – i.e., IF/THEN/ELSE, DO/ENDDO, etc. – to traditional code with labels is trivial.
  - In fact, that’s exactly what SPMs do!
- Converting unstructured code to structured code is an entirely different matter.
- Many custom constructs have no structured equivalent.
- So-called “spaghetti” code is especially difficult to convert.
- In some cases, the logic must be completely reworked to achieve a complete conversion.
- Fortunately, wholesale conversion is NEVER required!
Unstructured and Structured Code Coexistence in the Same Programs

- Structured and unstructured code easily coexist in the same program – even in the same routine.
- Our approach is to add new structured code and (mostly) leave existing unstructured code alone.
- If a routine requires a substantive rewrite, that becomes an opportunity for restructuring.
- There are always small restructuring opportunities that require almost no change. We do these when we can.
- Using this approach, frequently updated programs eventually become more structured than unstructured.
Taking Advantage of Small Restructuring Opportunities

• Every program has them – no matter how old or poorly written it might be.

• All labels ostensibly look the same. The more unimportant labels you can remove, the better the important labels, and therefore the important program flow, will stand out.

• Each label removed is one less potential branch target!

• After enough unimportant labels are removed, your mind’s pattern recognition abilities will eventually allow you to visualize how to restructure most unstructured routines.

• Start small; don’t rush things; work from inside-out.
Example #1 of a Small Restructuring Opportunity (Before and After)

- This code fragment is small and self contained.
- Label MCCON510 is referenced nowhere else and is removed with the conversion.
- Using IF this way, the tested condition must be inverted.

```
LA    R3, SUBSWKD1         Point to new console name
CLI   SUBSWKD1, C' '      Console name specified ?
JH    MCCON510            Branch if yes
XR    R3, R3              Zero out console name pointer
MCCON510 DS    0H
```

```
LA    R3, SUBSWKD1         Point to new console name
IF CLI, SUBSWKD1, LE, C' ' If no console name specified
XR    R3, R3              Zero out console name pointer
ENDIF ,                     EndIf
```
Example #2 of a Small Restructuring Opportunity (Before)

- This code fragment is small and self contained.
- Label MCCON520 is referenced nowhere else and is removed with the conversion.
- Set R4 to 1, -1 or zero.

```
LHI   R4,1                       Assume MIGID to be assigned
TM    SUBSFLG1,SUBS1MIG         MIGID to be assigned ?
JO    MCCON520                  Branch if yes
LNR   R4,R4                     Assume no MIGID to be assigned
TM    SUBSFLG1,SUBS1MNG         No MIGID to be assigned ?
JO    MCCON520                  Branch if yes
XR    R4,R4                     Allow default MIGID assignment

MCCON520  DS    0H

LHI   R5,1                       Indicate explicit activation
EJEESRV TYPE=MCSXACT,           Activate MCS Extended Console
   PARM=((R3),
   (R4),
   (R5)),
   TCBADDR=SUBSDU
J     MCCON900                  Return
```
Example #2 of a Small Restructuring Opportunity (After #1)

- The simplest and most direct conversion uses DO.
- ENDDO takes the place of label MCCON520.
- A “mindless” conversion; least risk of introduced error.

```
DO ,                          Do for MIGID parameter
  LHI  R4,1                    Assume MIGID to be assigned
  DOEXIT TM,SUBSFLG1,SUBS1MIG,O Exit if MIGID to be assigned
  LNR  R4,R4                   Assume no MIGID to be assigned
  DOEXIT TM,SUBSFLG1,SUBS1NMG,O Exit if no MIGID to be assigned
  XR   R4,R4                   Allow default MIGID assignment
ENDDO ,                       EndDo for MIGID parameter
  LHI  R5,1                    Indicate explicit activation
  EJESSRV TYPE=MCSXACT,        Activate MCS Extended Console
    PARM=((R3),             .. Console name
          (R4),             .. MIGID specification
          (R5)),            .. ACTIVATE request
          TCBADDR=SUBSDU          .. DU address (fullword)
  J    MCCON900                Return
```
Example #2 of a Small Restructuring Opportunity (After #2)

- An alternative solution using IF.
- The tested conditions must be inverted.

```assembly
LHI R4,1
IF TM,SUBSFLG1,SUBS1MIG,NO
  LNR R4,R4
  IF TM,SUBSFLG1,SUBS1MIG,NO
  XR R4,R4
  ENDIF,
ENDIF, 
LHI R5,1
EJESSRV TYPE=MCSXACT,
  PARM=((R3),
        (R4),
        (R5)),
        TCBADDR=SUBSDU
J MCCON900
```

Assume MIGID to be assigned
If no explicit MIGID assignment
Assume no MIGID to be assigned
If no explicit NOMIGID assignment
Allow default MIGID assignment
EndIf
EndIf no explicit MIGID assignment
Indicate explicit activation
Activate MCS Extended Console
  .. Console name
  .. MIGID specification
  .. ACTIVATE request
  .. DU address (fullword)
Return
Example #2 of a Small Restructuring Opportunity (After #3)

- An alternative solution using SELECT.
- Logic slightly changed but tested conditions are the same.
- Executes a little faster because R4 is updated only once.

```assembly
SELECT ,                     Select for MIGID parameter
    WHEN TM,SUBSFLG1,SUBS1MIG,O  When MIGID to be assigned
        LHI   R4,1               Show MIGID to be assigned
    WHEN TM,SUBSFLG1,SUBS1NMG,O  When no MIGID to be assigned
        LHI   R4,-1             Show no MIGID to be assigned
    OTHERWISE ,                   Otherwise use default
        XR    R4,R4             Allow default MIGID assignment
ENDSEL ,                      EndSel for MIGID parameter
    LHI   R5,1               Indicate explicit activation
EJESSRV TYPE=MCSXACT,       Activate MCS Extended Console
    PARM=((R3),             .. Console name
          (R4),             .. MIGID specification
          (R5)),            .. ACTIVATE request
    TCBADDR=SUBSDU          .. DU address (fullword)
J      MCCON900            Return
```
Example #3: This Routine Has Only One Label Now — An Obvious Conversion

**************************************************************
* Process CONSOLE Request                                      *
**************************************************************
MCCON500 DS 0H
   LTR  R5,R5               Anything specified?
   BZ   SUBSSE00              Branch if not
   TM   SUBSFLG1,SUBS1ACT          ACTIVATE requested?
   JZ   MCCON550              Branch if not
   EJESSRV TYPE=MCSXDAC,         Deactivate existing MCS console
      PARM=0,                    .. Implicit deactivate
      TCBADDR=SUBSDU             .. DU address (fullword)
   LA   R3,SUBSWKD1            Point to new console name
   IF   CLI,SUBSWKD1,LE,C' '   If no console name specified
      XR   R3,R3                   Zero out console name pointer
   ENDIF ,                      EndIf
   SELECT ,                     Select for MIGID parameter
      WHEN TM,SUBSFLG1,SUBS1MIG,O  When MIGID to be assigned
         LHI  R4,1                    Show MIGID to be assigned
      WHEN TM,SUBSFLG1,SUBS1NMG,O  When no MIGID to be assigned
         LHI  R4,-1                   Show no MIGID to be assigned
      OTHERWISE ,                    Otherwise use default
         XR   R4,R4                   Allow default MIGID assignment
   ENDSEL ,                      EndSel for MIGID parameter
   LHI   R5,1                    Indicate explicit activation
   EJESSRV TYPE=MCSXACT,        Activate MCS Extended Console
      PARM=((R3),                  .. Console name
         (R4),                     .. MIGID specification
         (R5)),                    .. ACTIVATE request
      TCBADDR=SUBSDU              .. DU address (fullword)
   J    MCCON900               Return
MCCON550 DS 0H
   EJESSRV TYPE=MCSXDAC,        Deactivate existing MCS console
      PARM=1,                    .. Explicit deactivate
      TCBADDR=SUBSDU              .. DU address (fullword)
   J    MCCON900               Return

Because we re-structured from inside-out, it’s easy to see what’s left to be done.
Example #3: After Conversion

***********************************************************************
* Process CONSOLE Request                                              *
***********************************************************************
MCCON500 DC 0H

LTR  R5,R5                  Anything specified ?
BZ   SUBSSE00                Branch if not

IF TM,SUBSFLG1,SUBS1ACT,NZ
  EJESSRV TYPE=MCSXDAC,         Deactivate existing MCS console
     PARM=0,
     TCBADDR=SUBSDU
  LA   R3,SUBSWKD1
  IF CLI,SUBSWKD1,LE,C' '       If no console name specified
     XR   R3,R3                   Zero out console name pointer
  ENDF ,                        EndIf
  SELECT ,                      Select for MIGID parameter
  WHEN TM,SUBSFLG1,SUBS1MIG,O  When MIGID to be assigned
     LHI  R4,1                    Show MIGID to be assigned
  WHEN TM,SUBSFLG1,SUBS1NMG,O  When no MIGID to be assigned
     LHI  R4,-1                   Show no MIGID to be assigned
  OTHRWISE ,                     Otherwise use default
     XR   R4,R4                   Allow default MIGID assignment
  ENDSSEL ,                      EndSel for MIGID parameter
     LHI  R5,1                    Indicate explicit activation
     EJESSRV TYPE=MCSXACT,         Activate MCS Extended Console
        PARM=((R3),
        (R4),
        (R5)),
        TCBADDR=SUBSDU
 ELSE ,                        Else DEACTIVATE requested
     EJESSRV TYPE=MCSXDAC,         Deactivate existing MCS console
        PARM=1,
        TCBADDR=SUBSDU
 ENDF ,                        EndIf ACTIVATE requested
J    MCCON900                Return
Example #3: The Final Implementation

***************************************************************
* Process CONSOLE Request                                     *
***************************************************************

MCNS500 DC 0H

   LTR R5, R5
   BZ SUBSSE00
   IF TM, SUBSFLG1, SUBS1ACT, Z
     EJESSRV TYPE=MCSXDAC,
       PARM=1,
       TCBADDR=SUBSDU
   ELSE ,
     EJESSRV TYPE=MCSXDAC,
       PARM=0,
       TCBADDR=SUBSDU
   LA R3, SUBSWKD1
   IF CLI, SUBSWKD1, LE, C','
     XR R3, R3
   ENDF ,
   SELECT ,
   WHEN TM, SUBSFLG1, SUBS1MIG, O
     LHI R4, 1
   WHEN TM, SUBSFLG1, SUBS1NMG, O
     LHI R4, -1
   OTHERWISE ,
     XR R4, R4
   ENDS ,
   LHI R5, 1
   EJESSRV TYPE=MCSXACT,
     PARM=((R3),
       (R4),
       (R5)),
     TCBADDR=SUBSDU
  ENDF ,
  J MCCON900

   Anything specified ?
   Branch if not
   IF DEACTIVATE requested
     Deactivate existing MCS console
     .. Explicit deactivate
     .. DU address (fullword)
   Else ACTIVATE requested
     Deactivate existing MCS console
     .. Implicit deactivate
     .. DU address (fullword)
     Point to new console name
     If no console name specified
     Zero out console name pointer
     EndIf
   Select for MIGID parameter
   When MIGID to be assigned
     Show MIGID to be assigned
   When no MIGID to be assigned
     Show no MIGID to be assigned
   Otherwise use default
   EndSel for MIGID parameter
   Indicate explicit activation
   Activate MCS Extended Console
     .. Console name
     .. MIGID specification
     .. ACTIVATE request
     .. DU address (fullword)
   EndIf DEACTIVATE requested
   Return

This last change was due only to my personal preference for a trivial THEN clause. It’s by no means needed.
Converting Some Code is Just Not Worth It – Comment it and Leave it Alone

************************************
* Loop for Each Block Remaining *
************************************
DO WHILE=(LTR,R15,R15,Z)  Do while more to process

************************************
* Read and Index Next Block  *
* (Not Structured Assembler) *
************************************
MVC   EPVFDB,DATNEXT      Move FDB of next block
LH    R4,EPVMRFIC         Get record count current block
MRFSX100   DS    0H
L     R1,EPVMRFBU        Point to disk buffer
JAS   R14,MRFSSPL        Read spool block
CHI   R15,4              Test return code
JH    MRFSX200           Branch if block not read
JE    *+12              Branch if VERIFY failed
JAS   R14,MRFSSVAL      /---Validate block just read
J     MRFSX100           0 Branch if DASD read required
JAS   R14,MRFBUF        4 /-Try to read storage buffer <---
J     MRFSX100           0 Branch if retry required
J     MRFSX200           4 Branch if validation failed
JAS   R14,MRFSSXBL      6 8 Extend valid spool block
JAS   R14,MRFSSIDX      /-- Build index for this block
J     MRFSX200           0 Branch if index error
LH    R4,EPVMRFIC        4 Get record count current block

************************************
* Set Return Codes from Read/Index *
* (Needed because those routines  *
* are not Structured Assembler.) *
************************************
XR    R15,R15            Set return code = 0
J     MRFSX300         Continue
MRFSX200   DS    0H
LHI   R15,4            Set return code = 4
MRFSX300   DS    0H

( the rest of the routine is 100% structured code)

ENDDO ,  EndDo while more to process

“Clever” use of vectored returns. Conversion not worth the hassle.
THE END