IBM Z System: Remediation of Programs with SIIS Impacts

Version Date: August 16, 2016

IBM Washington Systems Center

Kathy Walsh
Klaus Wolf

Assembler Programs

Common Examples:

Assembler Programs with a Store into Instruction Streams

This section of examples is intended to demonstrate some of the more commonly seen poor programming practices which result into program store interlock conditions. To be clear, these programming examples should be avoided not emulated.

Example 1 – In-stream Instruction Alteration

```
                LA      R3,15
                STC     R3,$MOVE+1
$MOVE           MVC     TARGET(0),SOURCE    perform a move in variable
*                                           length
```

This is building an instruction in the execution path. Rather than this technique it is preferred to use an EXecute instruction to do this operation. A better method:

```
                EX      R3,$MVC
                .
                .
                .
$MVC            MVC     TARGET(0),SOURCE
```

Figure 1. Simple Program Store

Example 2 - Embedded Save Areas

```
2F739000 ! 90EC D00C ! STM     R14,R12,X'C'(R13)    save caller's registers in caller's
savearea
2F739004 ! 182D      ! LR      R2,R13               R2=A(caller's_SA)
2F739006 ! 0700      ! BCR     X'0',0
2F739008 ! 4DD0 F07A ! BAS     R13,X'7A'(,R15)      R13=2F73900C and pass control to 2F73907A
*                                                   this BAS instruction defines A(current_SA)
*
*
*
*
2F73907A ! 50D2 0008 ! ST      R13,X'8'(R2)     save A(current_SA) in previous_SA
2F73907E ! 502D 0004 ! ST      R2,X'4'(R13)     save A(previous_SA) in current_SA
2F739082 ! 41C0 0FFF ! LA      R12,X'FFF'           define and load
2F739086 ! 41CC D001 ! LA      R12,X'1'(R12,R13)    R12 as second base register;
R12=R13+X'1000'
2F73908A ! 58F0 F090 ! L       R15,X'90'(,R15)      R15=AF739094
2F73908E ! 0BCF      ! BSM     R12,R15              enforce AMODE31 and continue at 2F739094
2F739090 ! AF73 9094 ! at 2F739090, we have: 2F739094 plus AMODE31 bit
*
2F739094 ! 41B0 0FFF ! LA      R11,X'FFF'           define and load
2F739098 ! 41BB C001 ! LA      R11,X'1'(R11,R12)    R11 as third base register; R11=R12+X'1000'
2F73909C ! 58F0 D098 ! L       R15,X'98'(,R13)
2F7390A0 ! 0BBF      ! BSM     R11,R15              continue at 2F7390A8
2F7390A4 ! AF73 90A8 ! at      2F7390A4, we have: 2F7390A8 plus AMODE31 bit
2F7390A8 ! 58F0 D0A4 ! L       R15,X'A4'(,R13)      R15=AF7390B4
2F7390AC ! 0B0F      ! BSM     0,R15                continue at 2F7390B4
2F7390B0 ! AF7390B4  ! at 2F7390B0, we have: 2F7390B4 plus AMODE31 bit
*
2F7390B4 ! 5010 C4A4 ! ST      R1,X'4A4'(,R12)      save caller's R1 at R12+4A4
```

In this example the routines housekeeping and save area chaining instructions and the (new/own) save area are interleaved within the same cache line. Changes have to be made.

Alternatives could be to move the save area into its own data area further away in the program, use GETMAIN/Storage Obtain to get a dedicated virtual storage area, or the save area needs to be restructured to have the save area defined in dedicated cache lines. Examples of how to do this are shown below.

Figure 2. Example of Embedded Save Area

Example 3 - Embedded Save Areas – Macro Driven

```
000000                00000 02CDC    16 IBMU050  CSECT ,         **** ENTRY NAME  ****
                                     19          XASTART R12,R11,SAVE=NP,MODE=(31,ANY)
                                     20+         PUSH  PRINT
000000                               21+         CNOP  0,4
                                     22+IBMU050  AMODE 31
                                     23+IBMU050  RMODE ANY
                                     24+         PRINT OFF
                                    297+         PRINT ON
                R:F   00000          298+         USING *,R15
000000 90EC D00C             0000C   299+         STM   R14,R12,X'C'(R13)   SAVE REGISTERS
000004 182D                          300+         LR    R2,R13    SAVE OLD SAVEA POINTER
000006 0700                          301+         CNOP  0,4
000008 4DD0 F07A             0007A   302+         BAS   R13,IBM1F  LOAD NEW SAVEA POINTER
00000C 0000000000000000              303+IBM1S    DC    18F'0'     NEW SAVE AREA
000054 D2E9C4E4F0F5F040              304+         DC    CL8'IBMU050'
00005C F1F061F2F661F1F5              305+         DC    C'10/26/15'
000064 F2F04BF3F2                    306+         DC    C'20.32'
000069 C7C1C440C8D6E4E2              307+         DC    C'PGM HOUSEKEEPING '
00007A 50D2 0008             00008   308+IBM1F    ST    R13,X'8'(R2)  STORE NEW SAVEA
00007E 502D 0004             00004   309+         ST    R2,X'4'(R13)  STORE OLD SAVEA POINTER
000082 41C0 0FFF             00FFF   310+         LA    R12,4095
000086 41CC D001             00001   311+         LA    R12,1(R12,R13)        LOAD NEXT BASE
```

In this example housekeeping and the save area chaining instructions and the (new/old) save area are interleaved within the same cache line. In this case the instructions are being caused by the macro XASTART. If routine IBMU050 passes control to a subordinate routine, then this subordinate routine will also cause an additional store into the instruction stream as soon as the called routine performs its save area chaining.

If the new save area to be defined must reside at the start of the current CSECT (i.e. close to the entry point), then this save area has to be placed into a dedicated data/operand cache line not containing any executable instruction. This objective will be achieved if the save area under consideration is defined as aligned to a 256-byte boundary within the CSECT and the CSECT itself is 4k page-aligned (e.g. via the PAGE BINDER / LINKAGE EDITOR control statement). In this case, the save area must be defined in a size of 256 bytes (or multiples of 256 bytes). This type of definition guarantees no executable code will be part of the data/operand cache line(s) containing such a save area.

If there is the option to relocate the save area under consideration to the end of the CSECT for addressability reasons, then this method should be used. In such a design, it is always possible to define a dummy area consisting of x‘100‘ bytes at the end of the executable code. This dummy area should cover the space between the last instruction of the CSECT and the beginning of the data/operand area of the CSECT. Good programming practices states the start of the data area is supposed to be aligned to a 256-byte boundary anyway. This recommendation assumes all Assembler CSECTs start at a 4k page boundary and consist of a “pure” instruction and a “pure” data/operand part and the data/operand section itself starts at a 256-byte boundary within the CSECT.

Figure 3. Embedded Save Area Macro Driven

Example 3A – Correction to XASTART macro

```
EDIT       SYS1.SYSMAC(XASTART) - 01.07                Columns 00001 00080
Command ===>                                           Scroll ===> CSR
000314           USING *,R15
000315 &NAME     STM   R14,R12,X'8'(R13)   SAVE REGISTERS
000316           LR    R2,R13     SAVE OLD SAVEA POINTER IN R2
000317           CNOP  0,4
000318           LA    R13,&GN.S  LOAD NEW SAVEA POINTER INTO R13
000319           B     &GN.F
000320           CNOP  0,4
000321 &SYMBDAT DC     C'&SYSDATE'
000322 &SYMBTIM DC     C'&SYSTIME'
000323           DC    C'PGM HOUSEKEEPING '
000324           AIF   (T'&SYSECT EQ 'O').A3
000325 &SYMBSCT DC     CL8'&SYSECT'
000326           AGO   .A4
000327 .A3       ANOP
000328 &SYMBSCT DC     CL8' '    NO SECTION-NAME
000329 .A4       ANOP
000330 O_INT_VM EQU    ((*-&SYSECT)/256+1)*256-(*-&SYSECT)
000331           ORG   *+O_INT_VM
000332 &GN.S    DC    64F'0'    NEW SAVEAREA
000333 &GN.F    ST    R13,X'8'(R2)  STORE NEW SAVEA
000334           ST    R2,X'4'(R13)  STORE OLD SAVEAPOINTER
000335           AIF   (N'&SYSLIST EQ 0).D1  NO MORE BASE
```

```
000000                 00000 02E64    16 IBMU050  CSECT ,                       **** ENTRY NAME
*
                                       19          XASTART R12,R11,SAVE=NP,MODE=(31,ANY)
                                       20+         PUSH  PRINT
000000                                 21+         CNOP  0,4
                                       22+IBMU050  AMODE 31
                                       23+IBMU050  RMODE ANY
                                       24+         PRINT OFF
                                      297+         PRINT ON
                 R:F  00000            298+         USING *,R15
000000 90EC D00C             0000C     299+         STM   R14,R12,X'C'(R13)   SAVE REGISTERS
000004 182D                            300+         LR    R2,R13     SAVE OLD SAVEA POINTER IN R2
000006 0700                            301+         CNOP  0,4
000008 41D0 F100             00100     302+         LA    R13,IBM1S  LOAD NEW SAVEA POINTER INTO
R13
00000C 47F0 F200             00200     303+         B     IBM1F
000010                                 304+         CNOP  0,4
000010 F1F161F0F261F1F5                305+         DC    C'11/02/15'
000018 F1F34BF3F7                      306+         DC    C'13.37'
00001D C7C1C440C8D6E4E2                307+         DC    C'PGM HOUSEKEEPING '
00002E D2E9C4E4F0F5F040                308+         DC    CL8'IBMU050'
                       000CA           309+O_INT_VM EQU   ((*-IBMU050)/256+1)*256-(*-IBMU050)
000036                 00036 00100     310+         ORG   *+O_INT_VM
000100 0000000000000000                311+IBM1S    DC    64F'0'    NEW SAVEAREA
000200 50D2 0008             00008     312+IBM1F    ST    R13,X'8'(R2)  STORE NEW SAVEA
000204 502D 0004             00004     313+         ST    R2,X'4'(R13)  STORE OLD SAVEA POINTER
000208 41C0 0FFF             00FFF     314+         LA    R12,4095
00020C 41CC D001             00001     315+         LA    R12,1(R12,R13)
000210 58F0 F218             00218     316+         L     R15,IBM00011
000214 0BCF                            317+         BSM   R12,R15
```

Look at the ORG technique shown in this example. It can be used as an alignment service macro. Generally, it determines the length of the program to the current spot and then places the next statement on a cache line boundary.

By changing the macro to request 64 full words the save area is now 256 bytes long and constitutes its own cache line. The next instruction, the store of register 13, will start at x'200' and is not interleaved with data areas.

Figure 4. Correction of Macro with Embedded Save Area

Use of AMASPZAP Utility to Detect Embedded Save Areas

By using the AMASPZAP utility you can inspect CSECTs which have potential stores into the instruction stream coming from embedded save areas. A cache line is x'100' bytes long and you can look for indicators of an imbedded save area. An example is shown below.

```
AMASPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
 DUMPT IBMU050 ALL

**CCHHR- 0E76000416   RECORD LENGTH- 002CE0         MEMBER NAME  IBMU050   CSECT NAME  IBMU050
 000000   90EC D00C  182D 0700  4DD0 F07A  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000
          STM  TRTR  LR   BCR   BAS  SRP
 000020   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

 000040   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  D2E9 C4E4  F0F5 F040  F1F0 61F2
                                                                    MVC  LGHRL SRP  SRP   MVO
 000060   F661 F1F5  F2F0 4BF3  F2C7 C1C4  40C8 D6E4     E2C5 D2C5  C5D7 C9D5  C740 50D2  0008 502D
               MVO   PACK SH    PACK       STH  OC       UNPKU MVC  BPRP       BPP  ST          ST
 000080   0004 41C0  0FFF 41CC  D001 58F0  F090 0BCF     8000 0094  41B0 0FFF  41BB C001  58F0 D098
               LA    CLCL LA    TRTR L     SRP  BSM      SSM        LA   CLCL  LA   LGFI  L    TRTR
 0000A0   0BBF 0000  8000 00A8  58F0 D0A4  0B0F 0000     8000 00B4  5010 C584  47F0 D0F2  47F0 D0D2
          BSM       SSM       L    TRTR  BSM           SSM        ST   BPRP  BC   TRTR  BC   TRTR
 0000C0   95F1 302E  4770 D0D2  5810 C584  5801 0008     4110 0B7C  58E0 C25C  41F0 0B7C  0E0E 1BFF
          CLI  LPER  BC   TRTR  L    BPRP  L            LA   BSM   L    CGFI  LA   BSM   MVCL SR
 0000E0   4700 0000  58E0 D0E0  0B0E 0000  8000 00F0     58DD 0004  50FD 0010  98EC D00C  0B0E 5810
          BC        L    TRTR  BSM       SSM           L         ST         LM   TRTR  BSM  L
 000100   C584 9180  1000 4780  D10E D247  DF8C C00C     D203 DF88  C260 47F0  DDD4 58A1  0000 D201
```

This x'100' bytes shown above has 18 fullwords without any instruction in them which is the typical size of a save area. This example matches the code segment seen on Figure 3. Embedded Save Area Macro Driven. In the example below the output of the utility is shown with the corrections as seen in Figure 4. Correction of Macro with Embedded Save Area.

```
AMASPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
 DUMPT  IBMU050   ALL

**CCHHR- 0E76000A11   RECORD LENGTH- 002E68         MEMBER NAME  IBMU050   CSECT NAME  IBMU050
 000000   90EC D00C  182D 0700  41D0 F100  47F0 F200     F1F1 61F0  F261 F1F5  F1F4 4BF3  F3D7 C7D4   *.........1..02.*
          STM  TRTR  LR   BCR   LA   MVO   BC   PACK     MVO        PACK MVO   MVO  SH    UNPK        *11/02/1514.33PGM*
 000020   40C8 D6E4  E2C5 D2C5  C5D7 C9D5  C740 C9E9     D4E4 F0F5  F040 0000  0000 0000  0000 0000   * HOUSEKEEPING IB*
          STH  OC    UNPKU MVC  BPRP       BPP                      SRP                    *MU050 ..........*
 000040   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
 000060   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
 000080   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
 0000A0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
 0000C0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
 0000E0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000   *................*
                                                                                                       *................*
```

Above is the first cache line and it contains the first four instructions and the DC entries.

```
000100   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

000120   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

000140   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

000160   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

000180   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

0001A0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

0001C0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000

0001E0   0000 0000  0000 0000  0000 0000  0000 0000     0000 0000  0000 0000  0000 0000  0000 0000
```

The second cache line starting at x'100' is all zeros. There is no executable code interspersed.

```
000200   50D2 0008  502D 0004  41C0 0FFF  41CC D001     58F0 F218  0BCF 0000  8000 021C  41B0 0FFF
         ST         ST         LA   CLCL  LA   TRTR     L    PACK  BSM        SSM        LA   CLCL
000220   41BB C001  58F0 D12C  0BBF 0000  8000 0230     58F0 D138  0B0F 0000  8000 023C  5010 C618
         LA   LGFI  L    MVN   BSM        SSM           L    MVN   BSM        SSM        ST   CGRL
000240   47F0 D186  47F0 D166  95F1 302E  4770 D166     5810 C618  5801 0008  4110 0B7C  58E0 C2F0
         BC   MVN   BC   MVN   CLI  LPER  BC   MVN      L    CGRL  L          LA   BSM   L    MSGFI
000260   41F0 0B7C  0E0E 1BFF  4700 0000  58E0 D174     0B0E 0000  8000 0278  58DD 0004  50FD 0010
         LA   BSM   MVCL SR    BC         L    MVN      BSM        SSM        L          ST
000280   98EC D00C  0B0E 5810  C618 9180  1000 4780     D1A2 D247  C020 C0A0  D203 C01C  C2F4 47F0
         LM   TRTR  BSM  L     CGRL TM    LPR  BC       MVN  MVC   LARL LARL  MVC  OIHF SLGFI BC
0002A0   DE68 58A1  0000 D201  A002 C39C  5810 C618     5831 0004  D200 C667  302E 95F0  C667 4780
         ED   L           MVC              L    CGRL     L          MVC  CLHRL LPER CLI  CLHRL BC
0002C0   D1F2 95F1  C667 4770  D1E2 9180  1004 47E0     D1F2 D247  C020 C0E8  D203 C01C  C2F8 47F0
         MVN  CLI   CLHRL BC    MVN  TM    LPR  BC       MVN  MVC   LARL IIHF  MVC  OIHF  AGFI BC
0002E0   DE68 D247  C020 C130  D203 C01C  C2FC 47F0     DE68 D201  C665 3000  D207 3026  C178 D201
         ED   MVC   LARL        MVC  OIHF  CGFI BC       ED   MVC   CHRL LPER  MVC  LPER        MVC
000300   A002 C39E  95F0 C665  4740 D144  95F9 C665     4720 D144  95F0 C666  4740 D144  95F9 C666
                    CLI  CHRL  BC   MVN   CLI  CHRL     BC   MVN   CLI  CLGHR BC   MVN   CLI  CLGHR
000320   4720 D144  D207 3026  C180 D201  A002 C3A0     D501 C665  C3A2 47D0  D248 D501  C665 C3A4
         BC   MVN   MVC  LPER        MVC               CLC  CHRL        BC    MVC  CLC   CHRL
```

The third cache line starting at x'200' contains instructions.

Example 4 – Modifying code on entry to a subroutine

```
READ     GET   (R2)
         LR    R5,R1
         CLI   SM101RTY,X'65'
         BNE   READ
         CLC   SM101STF(2),=X'0001'    subtype=0001 -> EDT101S1
         BE    CALL1011    invoke EDT101S1
         OC    SM101STF(2),SM101STF
         BNZ   READ        subtype=0   101 record is processed
*
         LA    R6,SM101TME
         LA    R7,BUFFER
         BAL   R14,DATTIM
         BAL   R14,PROCESS
         B     READ
*-------------------------------------------------------------------*
         DS    F
PROCESS  ST    R14,*-4
         MVC   BUFFER+23(4),SM101SID
         MVC   BUFFER+28(4),SM101SSI
         LA    R4,SM101END
         USING QWA0,R4
         BAL   R14,TRIPLET
         PUT   (R3),BUFFER    eject first record
         BAL   R14,CLEAR
*        format QWHS
         BAL   R14,FMTPSO     format product section IFCID=0003
*        format QWAC
         BAL   R14,FMTQWAC    format DSNDQWAC
         BAL   R14,FMTQXST    format DSNDQXST SQL accounting
         L     R14,PROCESS-4
         BR    R14
*-------------------------------------------------------------------*
```

This is clearly not re-entrant code and there are actually two problems with this code sequence. The store of register 14 is clearly a problem but the Load of Register 14 from PROCESS-4 and the subsequent BR R14 instruction will also cause a SIIS event.

Example 5 – Inline Macro Expansion

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                HLASM
R6.0
0002EE 50E0 B4AC            004AC   380 ENVIR    ST    R14,SAV004
                                    381          TIME
DEC,$TIME,LINKAGE=SYSTEM,DATETYPE=MMDDYYYY
                                    382+*        MACDATE  02/15/04
0002F2                              384+         DS    0H
0002F2 5110 0000            00000   385+         LAE   1,0(0,0)           ZERO ACCESS REGISTER
1
0002F6 4110 B300            00300   386+         LA    1,IHB0023D        POINT TO PARAMETER
LIST
0002FA 47F0 B31C            0031C   387+         B     IHB0023W          BRANCH AROUND LIST
000300                              388+IHB0023D DS    0F
000300 0000                         389+         DC    FL2'0'
000302 00                           390+         DC    FL1'0'
000303 00                           391+         DC    FL1'0'
000304 0000000000000000             392+         DC    2F'0'
00030C 0000000000000000             393+         DC    2F'0'
000314 0000000000000000             394+         DC    2F'0'
00031C                              395+IHB0023W DS    0H
00031C 9202 1003      00003         396+         MVI   3(1),2              TIME FLAGS INTO P
LIST
000320 9201 1002      00002         397+         MVI   2(1),1              DATE FLAGS INTO P
LIST
000324 58E0 0010            00010   398+         L     14,16(0,0)          CVT ADDRESS
000328 58EE 0304            00304   399+         L     14,772(14,0)        SFT ADDRESS
00032C 58EE 0130            00130   400+         L     14,304(14,0)        LX/EX FOR SERVICE
ROUTINE
000330 B218 E000      00000         401+         PC    0(14)
000334 51E0 B438            00438   402+         LAE   14,$TIME            ADDRESS OF USER'S
AREA
000338 D20F E000 100C 00000 0000C   403+         MVC   0(16,14),12(1)      MOVE OUTPUT FROM PARM
LIST
                                    404+*                                  TO THE USER'S AREA
```

The Load Address instruction at offset x'2F6' points register 1 to the inline parameter list. The code execution branches around the inline parameter list and then does two Move Immediate instructions into the parameter list. This is clearly a store into the instruction stream. One way to correct this technique is to use the List and Execute forms for the macro as seen below.

```
                                    379 *-------------------------------------------------------------------*
0002EE 50E0 B48C            0048C   380 ENVIR    ST    R14,SAV004
                                    381          TIME
DEC,$TIME,LINKAGE=SYSTEM,DATETYPE=MMDDYYYY,MF=(E,TIME$)
                                    382+*        MACDATE  02/15/04
0002F2 5110 B640            00640   384+         LAE   1,TIME$               PARAMETER LIST ADDRESS
0002F6 D71B 1000 1000 00000 00000   385+         XC    0(28,1),0(1)        CLEAR PARAMETER LIST
0002FC 9202 1003      00003         386+         MVI   3(1),2              TIME FLAGS INTO P LIST
000300 9201 1002      00002         387+         MVI   2(1),1              DATE FLAGS INTO P LIST
000304 58E0 0010            00010   388+         L     14,16(0,0)          CVT ADDRESS
000308 58EE 0304            00304   389+         L     14,772(14,0)        SFT ADDRESS
00030C 58EE 0130            00130   390+         L     14,304(14,0)        LX/EX FOR SERVICE ROUTINE
000310 B218 E000      00000         391+         PC    0(14)
000314 51E0 B418            00418   392+         LAE   14,$TIME            ADDRESS OF USER'S AREA
000318 D20F E000 100C 00000 0000C   393+         MVC   0(16,14),12(1)      MOVE OUTPUT FROM PARM LIST
000574 4040404040404040             589 BUFFER   DC    CL150' '
00060A                              590 HELP     DS    CL16
                                    591 *-------------------------------------------------------------------*
000620                              592 START    DS    D
000628                              593 STOP     DS    D
000630                              594 TMUSED_A DS    D
000638                              595 TMUSED_B DS    D
                                    596 *-------------------------------------------------------------------*
                                    597 TIME$    TIME  LINKAGE=SYSTEM,MF=L
                                    598+*        MACDATE  02/15/04
000640                              600+TIME$    DS    0F
000640                              601+         DS    XL28
000660                              602          LTORG
```

This correction moves the work areas away from the executable code.

General Coding Guidelines

The following are general coding guidelines which should be followed to write well-formed Assembler code.

- Don't mix or interleave in instructions / executable code any data or operands which are the target of store or update operations.

- Generate dummy areas of separation in a sufficient length at those locations where instruction and data/operands are adjacent to each other. An Assembler statement of the type:

  ```
  Label    DC    XL256'00'
  ```

  can be used to brute force this objective. Of course, there are smarter ways to separate Instruction cache lines from data cache lines based on knowledge of the cache line boundaries.

- Guarantee the 4k page alignment of any Assembler load module via the BINDER/LINKAGE Editor control statement PAGE:

  ```
  //LKED.SYSIN     DD *
         PAGE   DRIVER
         NAME   DRIVER(R)
  ```

- Where possible write to a re-entrant standard.

Fortran Programs

Compiled programs typically don't exhibit SIIS conditions because the compiler knows to avoid the poor programming practice. There are cases where a compiled program in a higher-level language, like Fortran, can mistakenly generate code which forces a SIIS condition.

The IBM Fortran compiler took an APAR in 2002 to correct a SIIS situation where the work areas were within 256 bytes of executable code. The APAR, PQ66981, corrected the situation by introducing a new option called WORKFILL. By compiling the FORTRAN code with WORKFILL=256, the writable data is separated from instructions on a 256- byte boundary. This remediation will remove the SIIS condition.

COBOL V4.2 Programs

The only SIIS issue reported to date in COBOL code is in COBOL V4.2 production code compiled using the debugging options (Test / NO Optimization) rather than the recommended (NOTEST/ FULL OPTIMIZATION) options. The use of the TEST option is for debugging purposes and should be carefully used at it will cause higher CPU consumption for the programs specifying this option.

Special Notices

This publication is intended to help the customer manage a migration to an IBM z Systems processor. The information in this publication is not intended as the specification of any programming interfaces provided by the IBM z Systems processors. See the publication section of the IBM programming announcement for the appropriate product for more information about what publications are considered to be product documentation. Where possible it is recommended to follow-up with product related publications to understand the specific impact of the information documented in this publication.

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Performance data contained in this document was determined in a controlled environment; therefore, the results which may be obtained in other operating environments may vary significantly. No commitment as to your ability to obtain comparable results is any way intended or made by this release of information.