Tutorial on Trimodal Programming on z/OS

SHARE 118 in Atlanta, Session 10409

Avri J. Adleman, IBM
adleman@us.ibm.com

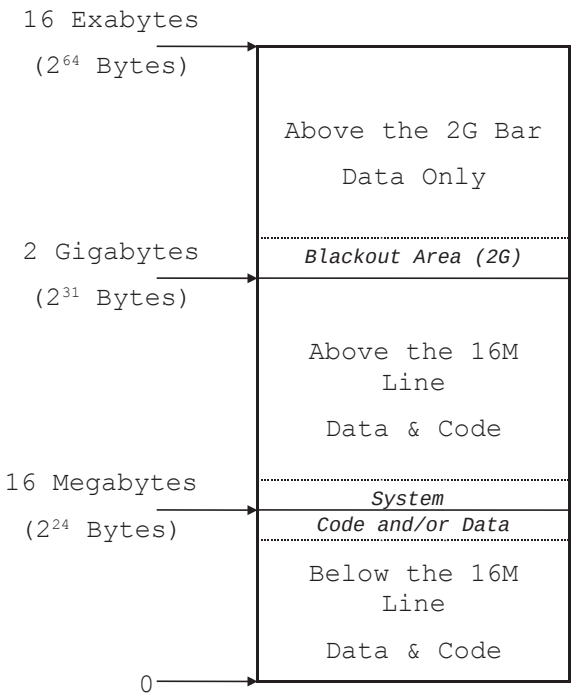
(Presented by John Ehrman, IBM)
March 16, 2012

Historical perspective

- Classical S360/370
  - (1964) Only 24-bit addressing supported
- 370/XA
  - (1981) Bimodal addressing: 24- or 31-bit addressing
- ESA/390
  - (1988) Dataspaces and Access Registers
- z/Architecture
  - (2001) Trimodal addressing: 24-, 31-, or 64-bit addressing

Memory layout

- Below the 16M Line
  - 24 Bit Addressing
  - Code: RMODE=24
  - Data: GETMAIN LOC=BELOW
- Above the 16M Line
  - 31 Bit Address
  - Code: RMODE=ANY (*may be below the line*)
  - Data: GETMAIN LOC=ANY (*may be below the line*)
- Above the 2G Bar
  - 64 Bit Address
  - Code: None (*as yet*)
  - Data: IARV64



Terminology: all machine generations

Byte	8 bits
Halfword	2 Bytes (16 Bits)
Fullword (Word)	4 Bytes (32 Bits)
Doubleword	8 Bytes (64 Bits)
Quadword	**16 Bytes (128 Bits)**

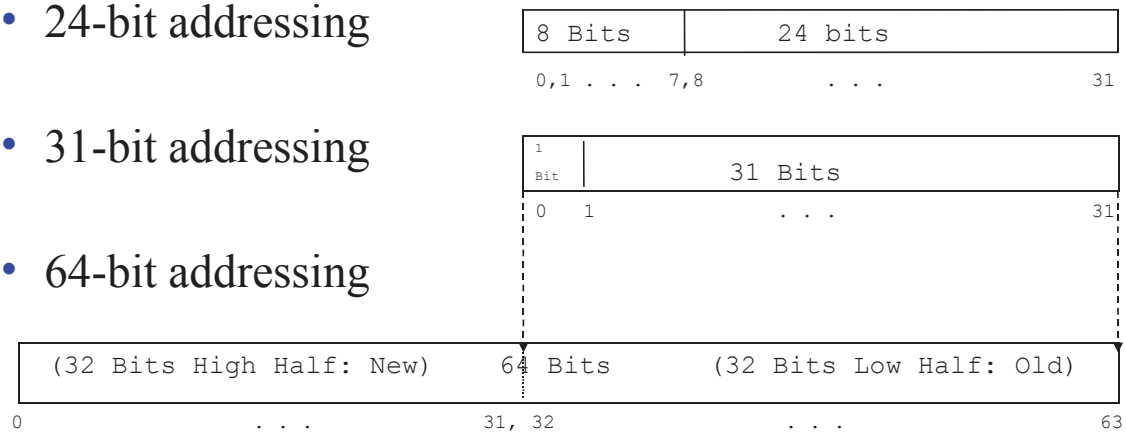
- **Notation:** *64-bit based* **[***32-bit based***]**
  - 64-bit based (Doubleword)
  - 32-bit based (Fullword)
- **Positions:**
  - "*High Order*" refers to the low numbered bits
  - "*Low Order*" refers to the high numbered bits

Registers

- 16 General Purpose Registers
  - In all generations of processors
- Pre-z/Architecture
  - 32 bits in size (the "*Traditional environment*")
- z/Architecture
  - 64 bits in size: (sometimes called "*Grande*")
  - Low Order Word: same as with Pre-z/Architecture processors
    - "*Traditional*" Instructions that use 24- and 31-bit addressing
  - High Order Word: z/Architecture Extension
    - New z/Architecture Instructions
      - Modal and modeless
      - 64 bit addressing
    - Ignored by "*traditional*" modeless instructions

Address formats

- 24-bit addressing
- 31-bit addressing
- 64-bit addressing



24-bit addressing in System/360

- General Purpose Registers
  - 32 bits
    - High 8 bits (0 to 7) for user (flags, etc)
      - BALR instruction (ILC, CC, Program Mask)
      - DCB fields
    - Low 24 bits (8 to 31) for addressing
- Special addressing-mode instructions
  - None (none needed!)

24- and 31-bit addressing in 370/XA

- General Purpose Registers
  - 32 Bits
    - High Order bit indicates addressing mode
      - 0 for 24-bit addressing, 1 for 31-bit addressing
    - Bits 1 to 7 depend on addressing mode
      - Part of the address (31-bit addressing)
      - Flags, etc. (24-bit addressing)
- Special addressing-mode instructions
  - BSM (*Branch and Set Mode*)
  - BASSM (*Branch and Save and Set Mode*)
    - More about these two, later

64-, 31-, and 24-bit addressing in z/Architecture

- General Purpose Registers: 64 Bits (Doubleword)
  - 32 Bit Extension (High Order Word)
    - Part of Address, Data, or Unused
  - 32 Bit Original (Low Order Word)
    - Retains Addressing Methodology for 24 -and 31-bit processing
    - Low order Bit 63[31]
      - Considered part of address -or- 64-bit addressing-mode indicator!
- Special addressing-mode instructions
  - Traditional:
    - BASSM, BSM
  - New with z/Architecture:
    - SAM24, SAM31, SAM64 and TAM

PSW description: 2 architecture modes

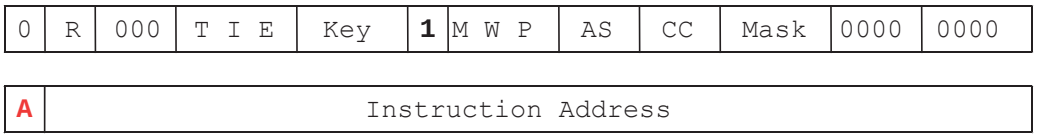
- ESA/390 mode
- Doubleword (64 Bits)
  - Bit 12 is always 1
  - Bit 31 is always 0
  - Instruction Address:
    - Bits 33 to 63
  - Addressing Mode (A)
    - Bit 32 determines addressing mode
      - 0 in 24-bit mode
      - 1 in 31-bit mode

- z/Architecture mode
- Quadword (128 Bits)
  - Bit 12 is always 0
  - Bit 31 contains the EA mode
  - Instruction Address
    - Bits 64 to 127
  - Addressing mode
    - Bit 31 (EA): Extended Addressing Mode
    - Bit 32 (BA): Basic Addressing Mode

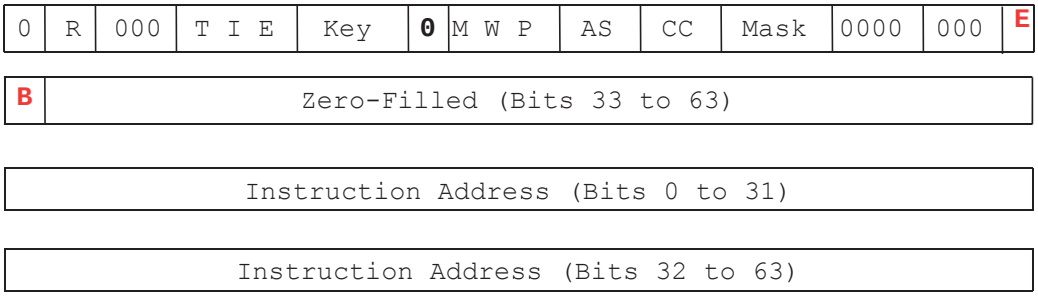
Addressing Modes		
	EA(0)	EA(1)
BA(0)	24	*Invalid*
BA(1)	31	64

PSW formats: 2 architecture modes

- ESA/390: Doubleword (64 bits)



- z/Architecture: Quadword (128 bits)



Architecture-mode-dependent instructions

- Processed differently based on Architecture Mode:
  - Same code may behave differently in z/Architecture mode vs. non- z/Architecture (ESA/390) mode
- Small (rare) number of cases
  - Examples:
    - BAKR and PR
      - Saves/Restores 64 bit registers
    - ESTA
      - PSW functions
    - BASSM & BSM
      - We will talk more about these two …
- Differences are minimal
- They do what you would expect

Modeless instructions

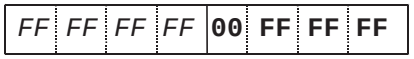
- Independent of architecture mode *and* addressing mode
  - Function is identical
    - Generally non-storage access type instructions
      - Register-register type instructions
      - Size of register access implied by instruction name
- General Purpose Registers
  - Pre-z/Architecture instructions
    - Operate only on low order word (bit 32[0] to bit 63[31])
    - High order word (bits 0 to 31 of 64) ignored
    - Examples: L, LR, A, AR, M, MR, SRDA, …
  - z/Architecture instructions
    - Operate either on 32-bit or all 64-bit registers
    - Examples: LGR (64-64), AGFR (64-32), RLL (32), RLLG (64), …

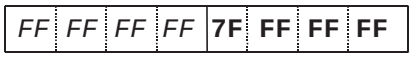
"Regular" modal instructions (1)

- Addresses function differently based on addressing mode
  - Base and Displacement are no different
  - May be hybrid with modeless
- Very predictable
  - No hidden surprises
- Generally the most commonly used instructions
  - Examples:
    - MVC – both storage operands depend on addressing mode
    - Loads and Stores (hybrids)
      - Arg_1 (register) size is based on instruction name (e.g. L vs LG)
      - Arg_2 (base and displacement) depends on addressing mode

"Regular" modal instructions (2)

- Load Address Types
  - LA, LAE, LAY: $R_1,D_2(X_2,B_2)$
  - Modal Processing – 64-bit register
    - 24-bit (high word is unchanged)
    - Low order word
      - Clears eight bits: 32[0] to 39[7]
      - Retains all other bits
    - 31-bit (high word is unchanged)
      - Low order word
      - Clears one bit: 32[0]
      - Retains all other bits
    - 64-bit
      - Sets full 64 bit register





- Lengths in registers usually interpreted based on addressing modes
  - Examples: CLCLE, MVCLE, TRE, … etc.
  - Some do not, such as MVCL and CLCL

"Irregular" modal instructions

- Function differently based on addressing mode
  - Reference or address storage
    - Base and Displacement
    - Register storage reference
- Have possible *"unpredictable"* side effects or processing
  - Visit: *Principles of Operations*
  - Read the fine print!
- Not many cases; usually, extensive or complex instructions
  - Examples:
    - TRT sets GPR 1 differently depending on addressing mode
    - ESTA (see code 1)
      - If possible use code 4

Mode-switching instructions

- Branch & Set Mode
  - BSM R_1,R_2
    - RR-Format:

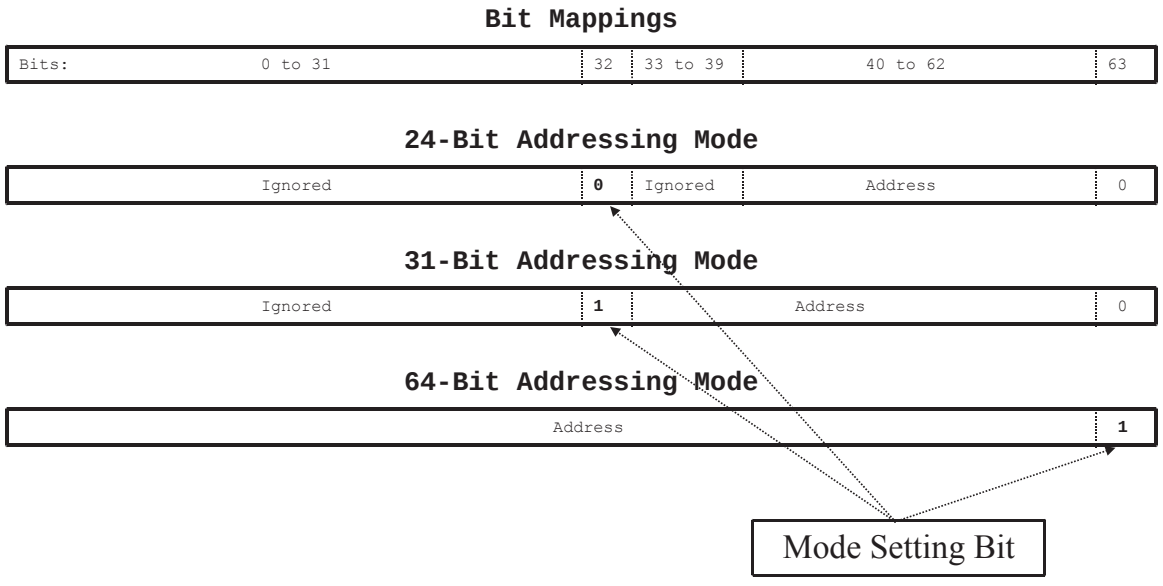
0B	R_1	R_2

  - R_1 = 32- or 64-bit register
    - $R_1 \neq 0$
      - Receives PSW addressing mode bit only; rest unchanged
    - $R_1 = 0$
      - No Address mode bit saved
  - R_2 = 32- or 64-bit register
    - $R_2 \neq 0$
      - Branch-to address
      - New addressing mode
    - $R_2 = 0$
      - No Branching
      - No Address Mode bit saved
- Branch & Save & Set Mode
  - BASSM R_1,R_2
    - RR-Format:

0C	R_1	R_2

  - R_1 = 32- or 64-bit register
    - R_1 any register number
      - Receives current PSW address (of next instruction) and PSW addressing mode
  - R_2 = 32- or 64-bit register
    - $R_2 \neq 0$
      - Branch-to address
      - New addressing mode
    - $R_2 = 0$
      - No Branching
      - No Address Mode bit saved

Register addressing-mode formats for BSM and BASSM



BASSM & BSM addressing

- Pre z/Architecture
  - Always an even branch address
  - 32-bit Register Only
  - High order bit mode setting
    - 0 for 24-bit addressing
    - 1 for 31-bit addressing
  - Low order bit
    - Part of instruction address
      - 0: Valid
      - 1: Odd instruction address; Invalid!

Note !!!

- z/Architecture 64-bit
  - ***Always*** an even branch address
  - 64-bit register
    - Low-order bit (63)
    - Not used as part of branch address
      - 0 for 24-, 31-bit addressing
      - 1 for 64-bit addressing
    - High/Low order bit 32[0]
      - For 24-, 31-bit addressing determines mode as in pre-z/Architecture
      - For 64-bit mode, part of instruction address

Mode-switching examples: calls within a single assembly

```
* Goto 24-bit mode from any mode
    LA       R15,GOTO24
    BASSM    R14,R15
```

```
* Goto 31-bit mode from any mode (1)
    L        R15,GOTO31@
    BASSM    R14,R15
* Goto 31-bit mode from any mode (2)
    LARL     R15,GOTO31
    OILH     R15,X'8000'
    BASSM    R14,R15
```

```
* Goto 64-bit mode from any mode
    XGR      R15,R15  For 24/31->64
    LARL     R15,GOTO64
    OILL     R15,X'0001'
    BASSM    R14,R15
```

```
GOTO31@  DC   A(GOTO31+X'80000000')
```

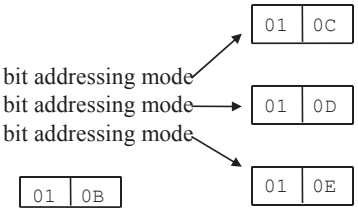
```
*  Entry into 24-bit Mode
GOTO24   DC    0H       Below the line
         . . .
         BSM   0,R14    Return to Caller
```

```
*  Entry into 31-bit Mode
GOTO31   DC    0H       Below/Above line
         . . .
         BSM   0,R14    Return to Caller
```

```
*  Entry into 64-bit Mode
GOTO64   DC    0H       Below/Above line
         . . .
         BSM   0,R14    Return to Caller
```

z/Architecture addressing-mode instructions

- SAM*xx:*
- Set Addressing Mode
  - E-Type (2-Byte) format with Opcode X'010x'
    - No registers set/modified; no register preload required
    - Old mode not retained
  - Types
    - SAM24: Switch to 24 bit addressing mode
    - SAM31: Switch to 31 bit addressing mode
    - SAM64: Switch to 64 bit addressing mode



- TAM
  - Test Addressing Mode

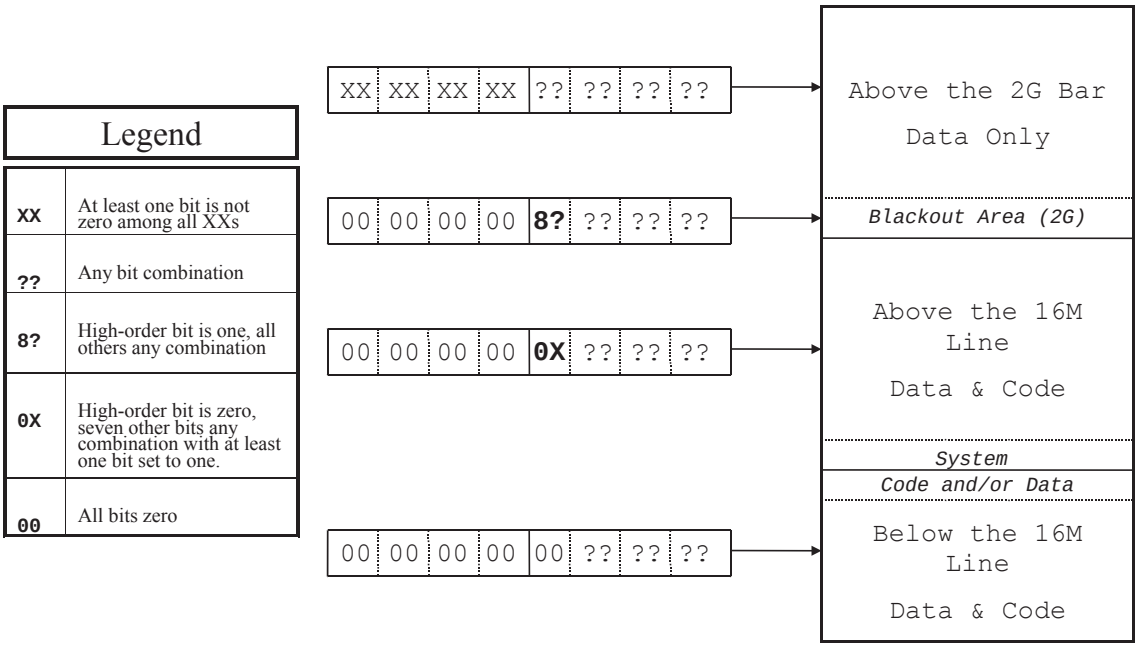
  - E-Type (2-byte) format:
  - Sets condition code based on current addressing mode
    - CC=0 – 24-bit addressing (Branch on Zero)
    - CC=1 – 31-bit addressing (Branch on Mixed)
    - CC=2 – Unused
    - CC=3 – 64-bit addressing (Branch on One)
  - No registers set or changed
  - Addressing mode is not switched

```
*  Examples:
       SAM24 , To AMODE(24)
       . . .
       SAM31 , To AMODE(31)
       . . .
       SAM64 , To AMODE(64)
       . . .
       TAM   , Test AMODE
       JZ    IN24
       JO    IN64
*  Running in AMODE(31)
IN31   DS    0H
       . . .
*  Running in AMODE(31)
IN24   DS    0H
       . . .
*  Running in AMODE(64)
IN64   DS    0H
       . . .
```


BALR vs. BASR

- BALR R_1,R_2
  - Since S/360
  - High order word
    - 24-, 31-bit mode: ignored
    - 64-bit mode: part of address
  - Processing Modes (R_1)
    - 24-bit addressing contains ILC, CC, Program Mask, 24 bit address
    - 31- and 64-bit addressing, identical to BASR
  - Deprecated now
    - Use BASR for branch and link
    - Use IPM instruction for CC and Program Mask
- BASR R_1,R_2
  - Since XA/370 (bimodal)
  - High order word
    - 24-, 31-bit mode: ignored
    - 64-bit mode: part of address
  - Processing Modes (R_1)
    - 24- and 31-bit addressing: contains mode bit and address
    - 64-bit addressing: contains only the address, no mode bit(s)
  - Preferred method of branch and link (or save) without mode switching
    - More consistent

Memory referencing by 64-bit addresses



AJA-23

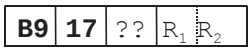
Special considerations

- 16M Line
  - System data and code straddles the line
  - Application code or data will not cross over
    - GETMAIN either totally below or above the line
    - Program object segment (class) will either be totally below or above the line
- 2G Bar
  - Blackout zone for MVS is first 2G above the 2G bar
    - **Technically, a valid addressable region!!**
    - Applies to a 64-bit address
      - High word is all zeroes
      - Low Order word has address with bit (32[0]) set to 1
    - IARV64 will not allocate storage in blackout zone

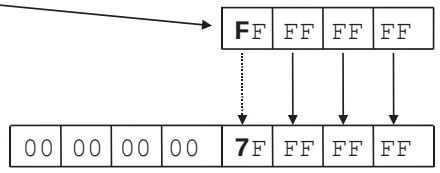
AJA-24

The useful LLGT and LLGTR instructions

- Load Logical *"Grande"* Thirty One Bits
  - **LLGT $R_1, D_2(X_2, B_2)$**
    - RXY Format: 

  - **LLGTR R_1, R_2**
    - RRE Format: 

- Source (Register or Storage)
  - Fullword, 32 bits (Arg_2)
- Target Register (R_1)
  - Doubleword, 64 bits
    - High word set to all zeroes
    - Low order word copied from source
    - Low order word, High Bit 32[0] set to 0



Example: Call and Return

```
          TITLE    'BAD CASE'
MYPGM     CSECT    ,
MYPGM     AMODE    MY_AMODE
MYPGM     RMODE    MY_RMODE
AMODE     EQU      ... bit setting ...
          . . .
          L        R15,YOURPGM@
          BASR     R14,R15
          . . .
YOURPGM@  DC       A(YOURPGM+AMODE)
          END      ,

YOURPGM   CSECT    ,
YOURPGM   AMODE    YOUR_AMODE
YOURPGM   RMODE    YOUR_RMODE
          . . .
          BSM      0,R14
          END      ,
```

```
          TITLE    'GOOD CASE'
MYPGM     CSECT    ,
MYPGM     AMODE    MY_AMODE
MYPGM     RMODE    MY_RMODE
AMODE     EQU      ... bit setting ...
          . . .
          XGR      R15,R15  <--Important!
          L        R15,YOURPGM@
          BASSM    R14,R15
          . . .
YOURPGM@  DC       A(YOURPGM+AMODE)
          END      ,

YOURPGM   CSECT    ,
YOURPGM   AMODE    YOUR_AMODE
YOURPGM   RMODE    YOUR_RMODE
          . . .
          BSM      0,R14
          END      ,
```

BAD CASE:	Worked OK for *MY_AMODE=YOUR_AMODE* for 24 and 31 but fails for 64
	Fails for *MY_AMODE≠YOUR_AMODE*
GOOD CASE:	Works for all *MY_AMODE* and *YOUR_AMODE* values

Notes: Call and Return

- Make sure CALL and RETURN types match
  - BASSM with BSM
  - BASR with BR
- Be sure alternatives are valid
  - LINK vs. LOAD and CALL
    - LINK: switches address mode as required
    - LOAD and CALL: does not switch addressing mode
- Watch out for addressing-mode bits as part of address
  - May have to clear address mode in register
    - Especially *"odd"* address and AMODE 64

Example: The KILLER bit!


```
         TITLE     'BAD PROGRAM'
MYPGM    CSECT     ,
MYPGM    AMODE     31
MYPGM    RMODE     ANY
         STM       R14,R12,12(R13)
         BASR      R11,0
         USING     *,R11
         . . .
         SAM64     ,
*****************************************
*  The Next Instruction Abends!!!       *
*  (because BASR executed in AMODE(31))  *
*****************************************
         MVC       DATA1,DATA2
         . . .
DATA1    DS        CL10
DATA2    DC        CL10'TESTING'
         . . .
         END       ,
```

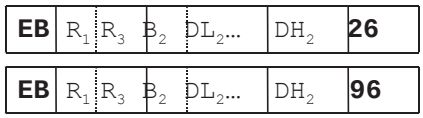
```
         TITLE     'GOOD PROGRAM'
MYPGM    CSECT     ,
MYPGM    AMODE     31
MYPGM    RMODE     ANY
         STM       R14,R12,12(R13)
         BASR      R11,0
         USING     *,R11
         . . .
         SAM64     ,
*****************************************
*  The Next Instruction Saves the Day   *
*  (Removes Blackout Area Addressing)   *
*****************************************
         LLGTR     R11,R11
         MVC       DATA1,DATA2
         . . .
DATA1    DS        CL10
DATA2    DC        CL10'TESTING'
         . . .
         END       ,
```

Linkage considerations

- New Instructions
  - Save and Load 64 bit registers
    - STMH, LMH, STG, LG, STMG, LMG, LMD
  - More on this follows …
- Save Areas
  - "Traditional" 72 byte save area
    - 32-bit registers
    - Standard Linkage
  - New Save Area Layout
    - 64-bit registers
    - Standard linkage
    - Transitional

Store/Load (Multiple) high halves of registers

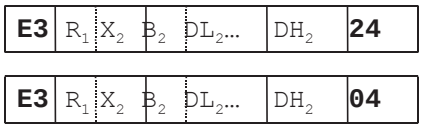
- Store/Load High Half of "*Grande*" Registers
  - Only high word's 32 bits saved
- Format RSY (extended displacement)
  - **STMH $R_1,R_3,D_2(B_2)$**
  - **LMH $R_1,R_3,D_2(B_2)$**



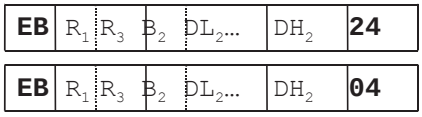
- Analogous to STM and LM
  - Acts on range of registers
    - No Store or Load instructions for high half of a single register
    - Use multiple-type instruction with $R_1 = R_3$

Store/Load entire 64-bit registers

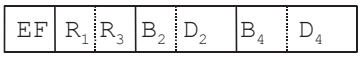
- STG and LG
  - Store and Load single 64-bit register
  - Analogous to ST (STY) and L (LY)
  - Format RXY:
    - **STG $R_1,D_2(X_2,B_2)$**
    - **LG $R_1,D_2(X_2,B_2)$**



- STMG and LMG
  - Store and Load multiple 64-bit registers
  - Analogous to STM (STMY) and LM (LMY)
  - Format RSY:
    - **STMG $R_1,R_3,D_2(B_2)$**
    - **LMG $R_1,R_3,D_2(B_2)$**



Load Multiple Disjoint

- **LMD $R_1,R_3,D_2(B_2),D_4(B_4)$**
  - Format SS: 

  - Loads range of full 64-bit registers
  - Uses two different locations
    - High half registers loaded from Arg_2
    - Low half registers loaded from Arg_4
    - Equivalent to doing a LMH and LM in one instruction!

```
*         Example of LMD
          STMH R2,R5,HIREGS
          STM  R2,R5,LOWREGS
          . . .
          LMD  R2,R5,HIREGS,LOWREGS
          . . .
HIREGS    DS 4F    Save High Half
LOWREGS   DS 4F    Save Low Half
```

- Allows AMODE=64 code to load saved "*Grande*" registers from two different save areas (high and low words)
  - Prevents register corruption on needed addresses
- Notes:
  - For performance, use sparingly:
    - Use LMH and LM or LMG if possible
  - There is **no** "*Store Multiple Disjoint*"

Save areas: old and new

- IHASAVER macro in SYS1.MACLIB
- Types of save areas:
  - "Traditional" 72 byte save area
    - 32-bit register save
    - Standard Linkage
  - Format 4
    - 64-bit register save
    - Standard linkage
      - Eye catcher "F4SA" at offset X'04'
      - Relocates "previous" and "next" chains to offset 128 (dec) and 136 (dec)
  - Format 5
    - 64-bit register save like format 4
    - 32-bit high register save area appended
      - Used for transition from 32 to 64 bit register processing
    - Standard linkage (like format 4)
      - Eye catcher "F5SA" at offset X'04'
      - Relocates "previous" and "next" chains to offset 128 (dec) and 136 (dec)

Save area layouts

- Traditional

+00	Reserved for Languages
+04	Previous Chain Pointer
+08	Next Chain Pointer
+0C	Return Address (R14)
+10	Entry Address (R15)
+14	GPR R0 to R12 Saved

Upper Half of Registers to "New" Save Area
Lower Half of Registers to "Old" Save Area

- z/Architecture 64-Bit

+00	Reserved for Languages
+04	**'F4SA'** or **'F5SA'**
+08	Return Address (G14)
+10	Entry Address (G15)
+18	GPR G0 to G12 Saved
+80	Previous Chain Pointer
+88	Next Chain Pointer
	F5SA Extension
+90	High Half Save of GPR G0 to G15

Sample program

- Two routines
  - **`Sample1`**
    - Runs either in 24- or 31-bit mode
    - Performs I/O to read and write records
    - Driver for a 64-bit-mode processing routine
  - **`RTN64`**
    - Entered in caller's mode (24 or 31)
      - Uses F5SA save area to save registers
    - Processes records in 64-bit mode
      - Allocates and deletes storage above the 2G bar
        - IARV64
      - Accesses storage above the 2G bar
        - Uses 64 bit registers

```
            PRINT NOGEN
SAMPLE1     CSECT ,
SAMPLE1     AMODE 31
SAMPLE1     RMODE 24
*********************************************************************
* PROGRAM SAMPLE1 - SAMPLE CALLING 64 BIT PROCESS ROUTINE FROM       *
*                   24/31 BIT PROGRAM                               *
*                                                                   *
*                   PROCESS IS NOT RENT                             *
*===================================================================*
* PROGRAM WRITTEN BY: AVRI J. ADLEMAN                               *
*                     ADLEMAN@US.IBM.COM                            *
*                     (732) 494-6093                                *
*===================================================================*
* PROCESS: SAMPLE1 READS IN AN 80 BYTE RECORD DATASET AND CALLS     *
*          THE RTN64 TO STORE THE RECORDS ABOVE THE 2G BAR.         *
*                                                                   *
*          AFTER ALL THE RECORDS ARE READ INTO RTN64 AREA, THE      *
*          RECORDS ARE THEN SORTED.                                 *
*                                                                   *
*          SAMPLE1 THEN RETRIEVES THE RECORDS, ONE AT A TIME AND    *
*          OUTPUTS THE IMAGE.                                       *
*                                                                   *
*          PROGRAM SAMPLE1 PERFORMS ALL I/O FUNCTIONS AND ACTS AS   *
*          A DRIVER TO THE RTN64 ROUTINE.  THE RTN64 ROUTINE PERFORMS *
*          A VARIETY OF PROCESSES ON THE ABOVE THE 2G BAR STORAGE   *
*          ARRAY.  THEY INCLUDE:                                    *
*                  (1) INITIALIZE THE ABOVE THE BAR REGION          *
*                  (2) TERMINATES (DELETES) THE ABOVE THE BAR REGION *
*                  (3) STORES RECORDS, ONE AT A TIME, SEQUENTIALLY  *
*                  (4) SORTS THE RECORDS                            *
*                  (5) RETURNS THE CURRENT RECORD COUNT             *
*                  (6) RETURNS ONE RECORD AT A TIME BY INDEX        *
*===================================================================*
* 32-BIT REGISTER DEFINITIONS                                       *
*********************************************************************
R0       EQU   0                       ----
R1       EQU   1                       PARMLIST
R2       EQU   2                       WORK REG
R3       EQU   3                       WORK REG
R4       EQU   4                       ----
R5       EQU   5                       ----
R6       EQU   6                       ----
R7       EQU   7                       ----
R8       EQU   8                       ----
R9       EQU   9                       ----
R10      EQU   10                      ----
R11      EQU   11                      PROGRAM BASE
R12      EQU   12                      ----
R13      EQU   13                      SAVE AREA ADDRESS
R14      EQU   14                      RETURN ADDRESS
R15      EQU   15                      ENTRY/RETURN CODE


*********************************************************************
*        SETUP STANDARD LINKAGE IN MAINLINE                         *
*********************************************************************
         SAVE  (14,12),,SAMPLE1-64-BIT-EXAMPLE-&SYSDATC-&SYSTIME
         BASR  R11,0                 SET R11 - PROGRAM BASE
         USING (*,ENDMAIN),R11       SETUP BASE
         ST    R13,SAVE+4            BACKWARD CHAIN
```

```
         LA    R8,SAVE                 R8 -> NEW SAVE AREA
         ST    R8,8(,R13)              FORWARD CHAIN
         LR    R13,R8                  R13 : CURRENT SAVE AREA
         OPEN  (INDCB,(INPUT),OUTDCB,(OUTPUT)),MODE=31
         LA    R1,INITPARM             R1  -> INITIALIZE PARM
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
*********************************************************************
*        OVERALL INPUT RECORDS STORE INTO ABOVE THE BAR ARRAY       *
*********************************************************************
READLOOP DS    0H
         GET   INDCB                   GET A RECORD
         ST    R1,ADDREC@              SAVE THE RECORD ADDRESS
         LA    R1,ADDPARM              R1  -> ADD RECORD PARMS
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
         J     READLOOP                GET NEXT RECORD
*********************************************************************
*        HIT THE END OF FILE - START PROCESSING                     *
*********************************************************************
EOF      DS    0H
*********************************************************************
*        SORT THE RECORDS                                           *
*********************************************************************
         LA    R1,SORTPARM             R1  -> INITIALIZE PARM
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
*********************************************************************
*        GET THE RECORD COUNT                                       *
*********************************************************************
         LA    R1,COUNTPRM             R1  -> INITIALIZE PARM
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
*********************************************************************
*        PUT OUT HEADER WITH RECORD COUNT                           *
*********************************************************************
         L     R5,COUNTNUM             R5 = RECORD COUNT
         CVD   R5,DWD                  MAKE DECIMAL IMAGE
         UNPK  HEADNUM,DWD             MAKE PICTURE
         MVZ   HEADNUM+L'HEADNUM-1(1),HEADNUM
         PUT   OUTDCB,HEADER           PUT HEADER TO OUTPUT


*********************************************************************
*        RETRIEVE AND PRINT ALL RECORDS                             *
*********************************************************************
         XR    R3,R3                   R3 = RECORD NUMBER
         LHI   R4,1                    R4 = INCREMENT
         AHI   R5,-1                   DECREMENT COUNT
         JM    DONE                    IF COUNT <= 0, DONE
RETRLOOP DS    0H
         ST    R3,RETRVNUM             SAVE NUMBER
         CVD   R3,DWD                  MAKE
         UNPK  DATANUM,DWD             MAKE PICTURE
         MVZ   DATANUM+L'DATANUM-1(1),DATANUM
         CVD   R3,DWD                  DWD = PICTURE OF COUNT
         LA    R1,RETRVPRM             R1  -> RETRIEVE PARM LIST
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
         PUT   OUTDCB,DATA             PUT RECORD TO OUTPUT
         JXLE  R3,R4,RETRLOOP          CONTINUE OVER ALL RECORDS
```

```
*********************************************************************
*        REMOVE THE ARRAY                                           *
*********************************************************************
DONE     DS    0H
         LA    R1,TERMPARM             R1  -> TERMINATION PARM
         LARL  R15,RTN64               R15 -> 64 BIT PROCESSOR
         BASR  R14,R15                 CALL ROUTINE
         CLOSE (INDCB,,OUTDCB),MODE=31
         FREEPOOL INDCB
         FREEPOOL OUTDCB
         L     R13,4(,R13)             R13 -> OLD SAVE
         L     R14,12(,R13)            R14 =  RETURN ADDRESS
         XR    R15,R15                 R15 =  0 - RETURN CODE
         LM    R0,R12,20(R13)          RESTORE R0 .. R12
         OI    15(R13),X'01'           MARK RETURNED
         BR    R14                     RETURN TO CALLER


*********************************************************************
*        MAINLINE STANDARD 72 BYTE SAVE AREA                        *
*********************************************************************
SAVE     DC    18F'0'
INITPARM DC    A(INIT#)
TERMPARM DC    A(TERM#)
SORTPARM DC    A(SORT#)
*
ADDPARM  DC    0A(0)
         DC    A(ADD#)
ADDREC@  DC    A(0)
*
COUNTPRM DC    0A(0)
         DC    A(COUNT#)
         DC    A(COUNTNUM)
*
RETRVPRM DC    0A(0)
         DC    A(RETRIEV#)
         DC    A(RETRVNUM)
         DC    A(DATAREC)
*
DWD      DC    D'0'
COUNTNUM DC    F'0'
RETRVNUM DC    F'0'
*
INIT#    DC    CL1'I'
TERM#    DC    CL1'T'
ADD#     DC    CL1'A'
SORT#    DC    CL1'S'
COUNT#   DC    CL1'C'
RETRIEV# DC    CL1'R'
*
HEADER   DC    CL133' '
         ORG   HEADER+1
         DC    C'NUMBER OF RECORDS: '
HEADNUM  DC    CL16' '
         ORG   ,
*
DATA     DC    CL133' '
         ORG   DATA+1
         DC    C'RECORD('
DATANUM  DC    CL16' '
         DC    C'): '
```

```
DATAREC  DC    CL80' '
         ORG   ,
*
         DC    0D'0',CL8'DCBS'
INDCB    DCB   DSORG=PS,MACRF=GL,LRECL=80,RECFM=FB,DDNAME=INFILE,      X
               DCBE=INDCBE
OUTDCB   DCB   DSORG=PS,MACRF=PM,LRECL=133,RECFM=FBA,DDNAME=OUTFILE,
X
               DCBE=OUTDCBE
INDCBE   DCBE  RMODE31=BUFF,EODAD=EOF
OUTDCBE  DCBE  RMODE31=BUFF
         LTORG ,
ENDMAIN  EQU   *


         DC    0D'0',CL8'RTN64'
RTN64    DC    0H'0'
*********************************************************************
* SUBROUTINE RTN64  - OPERATES IN 64 BIT MODE                       *
*                                                                   *
*            ENTRY VIA: BASR R14,R15 FROM 24 OR 31 BIT MODE         *
*                                                                   *
*            PARMS:     R1 -> PARMLIST (32-BIT)                     *
*                             (FUNCTION:CHAR(1),P#2, ... P#N)       *
*                                                                   *
*            ON EXIT:   R15 = RETURN CODE                           *
*                       REGISTERS R0 .. R12 - RESTORED              *
*                       R13 -> SAVE AREA SET/BIT(32) LOST           *
*                       R14 =  RETURN ADDRESS WITH MODE             *
*                                                                   *
* FUNCTION:                                                         *
* 'I'        -   INITIALIZE                                         *
* 'T'        -   TERMINATE                                          *
* 'S'        -   SORT                                               *
* 'A'        -   ADD RECORD                                         *
*                P#2 : INPUT_RECORD:CHAR(80)                        *
* 'C'        -   RETURN THE RECORD COUNT                            *
*                P#2 : RETURN THE RECORD COUNT:BINARY(31)           *
* 'R'        -   RETRIEVE THE ITH RECORD                            *
*                P#2 : RECORD_NUMBER:BINARY(31)                     *
*                P#3 : RETURNED_RECORD:CHAR(80)                     *
*===================================================================*
* 64-BIT REGISTER DEFINITIONS                                       *
*********************************************************************
G0       EQU   0                        ----
G1       EQU   1                        PARMLIST
G2       EQU   2                        WORK REG
G3       EQU   3                        WORK REG
G4       EQU   4                        WORK REG
G5       EQU   5                        WORK REG
G6       EQU   6                        WORK REG
G7       EQU   7                        WORK REG
G8       EQU   8                        ----
G9       EQU   9                        ----
G10      EQU   10                       PARMLIST REGISTER
G11      EQU   11                       STATIC PROGRAM BASE
G12      EQU   12                       ----
G13      EQU   13                       SAVE AREA ADDRESS
G14      EQU   14                       RETURN ADDRESS
G15      EQU   15                       ENTRY/RETURN CODE
*
```

```
RECSIZE  EQU   80                      SIZE OF A RECORD

*********************************************************************
*        START PROCESSING IN CALLER'S MODE                          *
*********************************************************************
OLDSAV13 USING SAVER,R13
         STM   R14,R12,OLDSAV13.SAVGRS14    SAVE REGISTERS R14 .. R12
         STMH  G15,G15,OLDSAV13.SAVNEXT     TEMP SAVE HIGH G15
         LARL  G15,SAVE64              G15 -> NEW SAVE AREA FORMAT 5
NEWSAV15 USING SAVF5SA,G15             G15 : TEMP BASE TO NEW SAVE
         STMH  G0,G14,NEWSAV15.SAVF5SAG64HS0  SAVE HIGH G0 .. G14
         MVC   NEWSAV15.SAVF5SAG64HS15,OLDSAV13.SAVNEXT
         ST    R15,OLDSAV13.SAVNEXT    SET FORWARD CHAIN IN OLD SAVE
         LLGTR G2,R13                  G2 -> OLD SAVE CLR HIGH ORDER
         STG   G2,NEWSAV15.SAVF5SAPREV SAVE 64-BIT PREVIOUS SAVE AREA
         LLGTR G13,G15                 G13-> NEW SAVE, CLEAR HIGH
         DROP  NEWSAV15,OLDSAV13       RELEASE TEMP SAVE BASE
CURSAV13 USING SAVF5SA,G13             G13 : CURRENT SAVE AREA BASE
*********************************************************************
*        SETUP ADDRESSABILITY AND PROCESS IN AMODE=64               *
*********************************************************************
         SAM64 ,                       GO TO 64 BIT MODE
         JAS   G11,PROCESS             G11 -> CURRENT ADDRESS
         USING (*,ENDRTN64),G11        G11 : PROGRAM BASE
CODE     LOCTR ,
*********************************************************************
*        GO TO FUNCTION REQUESTED                                   *
*********************************************************************
PROCESS  DS    0H
         LLGTR G10,R1                  G10 -> PARMLIST
         LLGT  G2,0(,G10)              FETCH FUNCTION CODE
         CLI   0(G2),C'A'              ADD A RECORD
         JE    ADD
         CLI   0(G2),C'R'              RETRIEVE A RECORD
         JE    RETRIEVE
         CLI   0(G2),C'C'              COUNT RECORDS
         JE    COUNT
         CLI   0(G2),C'S'              SORT RECORDS
         JE    SORT
         CLI   0(G2),C'I'              INITIALIZE REGION
         JE    INITIAL
         CLI   0(G2),C'T'              TERMINATE  REGION
         JE    TERMINAT
         LHI   R15,16                  UNKNOWN TYPE, RC = 16
         J     RETURN                  RETURN TO CALLER
*********************************************************************
*        INITIALIZE RECORD AREA                                     *
*********************************************************************
INITIAL  DS    0H
         XGR   G0,G0                   G0 = 0
         STG   G0,RECCNT               SET RECORD COUNT = 0
         IARV64 REQUEST=GETSTOR,SEGMENTS=SEGMENTS,                     X
               GUARDLOC=HIGH,GUARDSIZE=GRDSIZE,                        X
               RETCODE=RETCODE,RSNCODE=RSNCODE,ORIGIN=ORIGIN,          X
               MF=(E,IARV64)
         ICM   R15,15,RETCODE          R15 = RETURN CODE
         JZ    RETURN                  IF RC = 0, GOOD
         J     RC12                    ELSE RETURN TO CALLER

*********************************************************************
```

```
*        RETURN THE COUNT OF RECORDS                                    *
*********************************************************************
COUNT    DS    0H
         LLGT  G2,4(,G10)              G2 -> RETURN COUNT
         ICM   R3,15,RECCNT+4          R3 = LOWER PART OF COUNT
         ST    R3,0(,G2)               SAVE COUNT
         JM    RC08                    INDICATE COUNT OVERFLOW
         ICMH  G3,15,RECCNT            FETCH TOP HALF OF
         JZ    RC00                    IF ZERO, OK
         J     RC08                    ELSE COUNT OVERFLOW
*********************************************************************
*        ADD A RECORD TO THE ABOVE THE LINE ARRAY                   *
*********************************************************************
ADD      DS    0H
SAMRECG2 USING RECORD,G2
RTNRECG5 USING RECORD,G5
         LLGT  G2,4(,G10)              G2 -> INPUT RECORD
         LG    G3,RECCNT               G5 =  NUMBER OF RECORDS
         LG    G4,ORIGIN               G4 -> ORIGIN
         LGR   G5,G3                   G5 =  RECORD COUNT
         MGHI  G5,RECSIZE              G5 =  OFFSET
         LA    G5,0(G5,G4)             G5 -> RECORD SLOT
         MVC   RTNRECG5.CARD,SAMRECG2.CARD   MOVE IN RECORD
         AGHI  G3,1                    BUMP RECORD COUNT
         STG   G3,RECCNT               SAVE VALUE
         J     RC00                    RETURN TO CALLER
         DROP  RTNRECG5,SAMRECG2       RELEASE BASES
         PUSH  USING
*********************************************************************
*        SORT THE RECORDS                                           *
*********************************************************************
SORT     DS    0H
LEAST    USING RECORD,G2               G2 : LEAST RECORD
TOP      USING RECORD,G4               G4 : CURRENT TOP
CURRENT  USING RECORD,G5               G5 : CURRENT AT RECORD
         LG    G3,RECCNT               G3 =  NUMBER OF RECORD
         LTGR  G3,G3                   DO WE HAVE ANY ?
         JNP   RC04                    NO, RETURN
         LG    G4,ORIGIN               G4 -> ORIGIN
         LGHI  G6,RECSIZE              G6 =  SIZE OF RECORD
         BCTGR G3,0                    G3 =  RECORD COUNT - 1
         MSGR  G3,G6                   G3 =  MAXIMUM OFFSET
         LA    G7,TOP.RECORD(G3)       G7 -> LAST RECORD
*********************************************************************
*        OVERALL RECORDS, SET THE TOP FOR BUBBLE SORT               *
*********************************************************************
SORTLP1  DS    0H
         LGR   G2,G4                   G2 -> CURRENT LEAST
         LGR   G5,G4                   G5 -> START

*********************************************************************
*        LOOK FOR THE LEAST RECORD                                  *
*********************************************************************
SORTLP2  DS    0H
         CLC   CURRENT.CARD,LEAST.CARD TEST WITH LEAST
         JNL   SORTNX2                 IF STILL GREATER, SKIP
         LA    G2,CURRENT.RECORD       ELSE SET NEW LEAST
SORTNX2  DS    0H
         JXLEG G5,G6,SORTLP2           GO TO NEXT ENTRY
*********************************************************************
*        OPTIONALLY SWAP LEAST RECORD WITH CURRENT TOP              *
```

```
*********************************************************************
         LA    G0,LEAST.CARD           G6 -> LEAST RECORD
         LA    G1,TOP.CARD             G7 -> TOP RECORD
         CLGR  G0,G1                   ANY SWITCH ?
         JE    SORTNX1                 NO, KEEP ASIS
         XC    LEAST.CARD,TOP.CARD     ELSE SWAP RECORDS
         XC    TOP.CARD,LEAST.CARD     ...
         XC    LEAST.CARD,TOP.CARD     ...
*********************************************************************
*        GO TO NEXT TOP RECORD                                      *
*********************************************************************
SORTNX1  DS    0H
         JXLEG G4,G6,SORTLP1           GO TO NEXT ENTRY
         J     RC00                    SORT COMPLETE
         POP   USING

*********************************************************************
*        RETURN THE ITH RECORD (ORIGIN NUMBER = 0)                  *
*********************************************************************
RETRIEVE DS    0H
         LLGT  G2,4(,G10)              G2 -> RECORD NUMBER TO RETRIEVE
         LGF   G3,0(,G2)               G3 =  RECORD NUMBER
         LTGR  G3,G3                   TEST VALUE
         JM    RC08                    IF NEGATIVE, ERROR
         CG    G3,RECCNT               TEST WITH CURRENT NUMBER
         JNL   RC08                    IF HIGH, RANGE ERROR
         MGHI  G3,RECSIZE              G3 = OFFSET OF RECORD
         ALG   G3,ORIGIN               G3 -> RECORD
         LLGT  G2,8(,G10)              G2 -> TARGET
SAMRECG2 USING RECORD,G2
RTNRECG3 USING RECORD,G3
         MVC   SAMRECG2.CARD,RTNRECG3.CARD   COPY IN RECORD
         DROP  RTNRECG3,SAMRECG2
         J     RC00                    RETURN TO CALLER
*********************************************************************
*        TERMINATE - REMOVE THE ABOVE THE BAR ARRAY                 *
*********************************************************************
TERMINAT DS    0H
         IARV64 REQUEST=DETACH,MEMOBJSTART=ORIGIN,                    X
               RETCODE=RETCODE,RSNCODE=RSNCODE,MF=(E,IARV64)
         ICM   R15,15,RETCODE
         JZ    RETURN
         J     RC12

*********************************************************************
*        RETURN TO CALLER IN MODE                                   *
*********************************************************************
RC16     DS    0H
         LHI   R15,16
         J     RETURN
RC12     DS    0H
         LHI   R15,12
         J     RETURN
RC08     DS    0H
         LHI   R15,8
         J     RETURN
RC04     DS    0H
         LHI   R15,4
         J     RETURN
RC00     DS    0H
         XR    R15,R15
```

```
RETURN   DS    0H
         LG    G2,CURSAV13.SAVF5SAPREV G2 -> OLD SAVE AREA
         LA    G3,CURSAV13.SAVF5SA     G3 -> CURRENT SAVE
         DROP  CURSAV13                RELEASE SAVE AREA BASE
OLDSAV2  USING SAVER,R2                G2 : BASE TO PREVIOUS SAVE
CURSAV3  USING SAVF5SA,G3              G3 : CURRENT SAVE AREA BASE
         LMH   G13,G15,CURSAV3.SAVF5SAG64HS13  RESTORE HIGH G13 .. G15
         LR    R13,R2                  R13 -> PREVIOUS SAVE
         L     R14,OLDSAV2.SAVGRS14    G14 =  RETURN ADDRESS WITH MODE
         LMD   G0,G12,CURSAV3.SAVF5SAG64HS0,OLDSAV2.SAVGRS0
         DROP  CURSAV3,OLDSAV2
         BSM   0,G14                   RETURN TO CALLER

SAMPLE1  LOCTR ,
*************************************************************************
*        FORMAT 5 SAVE FOR RTN64                                        *
*************************************************************************
SAVE64   DC    XL(SAVF5SA_LEN)'00'
         ORG   SAVE64+(SAVF5SAID-SAVF5SA)
         DC    A(SAVF5SAID_VALUE)
         ORG   ,
SEGMENTS DC    FD'10'
GRDSIZE  DC    F'1'
RETCODE  DC    F'0'
RSNCODE  DC    F'0'
ORIGIN   DC    AD(0)
RECCNT   DC    FD'0'
         IARV64 MF=(L,IARV64)
         LTORG ,
ENDRTN64 EQU   *
*************************************************************************
*        RECORD IMAGE                                                   *
*************************************************************************
RECORD   DSECT ,
CARD     DS    CL(RECSIZE)
         PUSH  PRINT
         PRINT NOGEN
*************************************************************************
*        SYSTEM CONTROL BLOCKS                                          *
*************************************************************************
         IHASAVER ,
         DCBD  DEVD=DA,DSORG=PS
         POP   PRINT
         END   ,
```