

64-Bit z/OS Assembler Coding

SHARE 101 Washington, DC
Session 8158 August, 2003

David Bond
Tachyon Software LLC
dbond@tachyonsoft.com

© Tachyon Software LLC, 2002-2003

Permission is granted to SHARE Incorporated to publish this material for SHARE activities and for others to copy, reproduce, or republish this material in a manor consistent with SHARE's By-laws and Canons of Conduct. Tachyon Software retains the right to publish this material elsewhere.

Tachyon Software is a registered trademark and Tachyon z/Assembler and Tachyon Operating System are trademarks of Tachyon Software LLC.

IBM, HLASM, OS/390, z/OS, z/Architecture, zSeries and System/390 are trademarks or registered trademarks of the International Business Machines Corporation.

Other trademarks belong to their respective owners.

The code included in this presentation is for illustrative purposes only and has NOT been subjected to extensive testing. The code is presented WITHOUT ANY WARRANTY and without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

z/Architecture Overview

What is unchanged from ESA/390?

- Access Registers (still 16 32-bit registers)
- Floating Point Registers (still 16 64-bit registers)
- I/O (mostly)
- Interrupts (with a few new Program Checks)
- IPL (starts in ESA/390 mode)
- ESA/390 instructions operating in 24-bit and 31-bit addressing mode

As IBM customers have come to expect, major extensions have again been added to the architecture without impacting older programs. Most OS/360, MVS, MVS/XA and MVS/ESA programs will run correctly under z/OS in z/Architecture mode without recompiling or relinking.

z/Architecture Overview

What is changed from ESA/390?

- General Registers (changed from 32 to 64 bits)
- Control Registers (changed from 32 to 64 bits)
- PSW (changed from 64 to 128 bits + new format)
- PSA (changed from 4K to 8K + new format)
- Linkage Stack (new format to allow saving of 64-bit general registers and 128-bit PSW)
- Many new instructions
- Some ESA/390 instructions act differently in 64-bit addressing mode.
- No expanded storage or vector facility.

The PSW format was changed to allow instruction addresses above 2 Gigabytes. However, as long as the instruction address is below 2 Gigabytes, the 128-bit PSW can be mapped into the ESA/390 64-bit PSW with no loss of information.

The first 33 bits and the last 31 bits of the PSW are the same in both formats with the following exceptions:

Bit 12: 1 for 64-bit PSW (old S/370 EC-mode bit), 0 for 128-bit PSW.

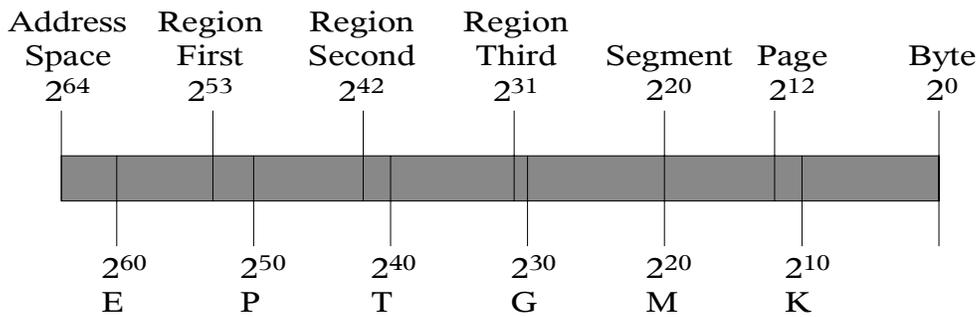
Bit 31: Always 0 in S/370 and ESA/390.

1 for AMODE 64, 0 for AMODE 24 or 31 in z/Architecture.

LPSW instruction automatically converts a 64-bit PSW to 128-bit format for z/Architecture. New LPSWE instruction is used to load a 128-bit format PSW.

z/Architecture Overview

Regions



Page	2^{12}	4K	Kilobyte	2^{10}	1,024
Segment	2^{20}	1M	Megabyte	2^{20}	1,048,576
Region 3rd	2^{31}	2G	Gigabyte	2^{30}	1,073,741,824
Region 2nd	2^{42}	4T	Terabyte	2^{40}	1,099,511,627,776
Region 1st	2^{53}	8P	Petabyte	2^{50}	1,125,899,906,842,624
Address Space	2^{64}	16E	Exabyte	2^{60}	1,152,921,504,606,846,976

z/Architecture Overview

ESA/390 instructions in 64-bit mode

Most ESA/390 instructions operate identically in 64-bit addressing mode except for register contents used for addresses. Non-address register references are unchanged. Examples:

- AR (add register) operates only on the lower 32-bit registers halves. The upper 32-bits of the registers are not referenced or changed.
- L (load) alters the lower 32 bits of the target register. However, all 64 bits of the base and index registers are used to generate the source address.
- MVCL (move long) still uses and alters 24 bits of the source and target length registers, but uses and alters all 64 bits of the address registers.

z/Architecture Principles of Operation calls MVCL a “modal instruction” because in AMODE 64, all 64 bits of the source and target registers are altered. However, MVCL is no more “modal” than L is since in AMODE 64 the L instruction uses all 64 bits of the base and index registers. The “modal instructions” listed in *Principles of Operation* are those which conditionally alter the upper register halves. For some reason, *Principles of Operation* does not include instructions that conditionally reference the upper register halves as “modal”.

z/Architecture Overview

ESA/390 instructions in 64-bit mode

- A few instructions use and update 64-bit length values when in 64-bit addressing mode: CLCLE, CLCLU, CUSE, CMPSC, CUUTF, CUTFU, MVCLE, MVCLU, MVCP, MVCS, MVCK, TRE, TROO, TROT, TRTO, TRTT.
- CFC and UPT operate quite differently in 64-bit mode.
- BASSM and BSM use/set low-order bit to indicate 64-bit mode.
- ESTA code 1 returns the 128-bit PSW mapped to a 64-bit PSW. Use new code 4 to get the whole PSW.
- LRA fails if the virtual storage is backed by real storage above 2G. Use LRAG or TPROT.

CLCLE, CLCLU, CUSE, CMPSC, CUUTF, CUTFU, MVCLE, MCVLU, MVCP, MVCS, MVCK, TRE, TROO, TROT, TRTO and TRTT are more “modal” than other instructions because in AMODE 64 all 64 bits of the length registers are used and updated.

CFC and UPT are the most modal because they act so differently in AMODE 64, processing 6 bytes at a time instead of 2 bytes.

BASSM and BSM examine and set the low order register bit to indicate AMODE 64. In ESA/390, BASSM or BSM to an odd address would cause a Program Check for Specification Exception. In z/Architecture mode, BASSM and BSM to an odd address will cause a switch to AMODE 64.

z/Architecture Overview

New Instructions

- z/Architecture introduces “G” (grande or große) versions of many instructions to explicitly operate on the full 64-bit register contents. Examples (64/32): LGR/LR, LG/L, SGR/SR, LMG/LM, AHGI/AHI, BCTG/BCT.
- New mixed 64/32-bit instructions - for example LGF: loads 32 bits sign-extended to 64 bits.
- New partial register instructions - examples are: NIHH,NIHL,NILH,NILL, LMH and ICMH.
- “Relative Long” instructions: LARL, BRCL, BRASL.
- New address mode instructions: SAM24, SAM31, SAM64, TAM.
- And many more...

Halfword Instructions:

And Immediate (NIxx)	xx: HH=bits 0-15
Insert Immediate (IIxx)	HL=bits 16-31
Load Logical Immediate (LLxx)	LH=bits 32-47
Or Immediate (OIxx)	LL=bits 48-63
Test Under Mask (TMxx)	

z/Architecture Overview

z990 Instructions

- The z990 processors support “long displacement” RX-format instructions, with a displacement field range of -524288 to +524287 ($\pm\frac{1}{2}$ Megabyte)
- All of the z/Architecture RX-format instructions introduced by the z900 are “long displacement” except for the floating point instructions.
- New “Y” (long displacement) forms of most of the ESA/390 RX-format instructions except branch and floating point. Examples: AY, AHY, LY, LAY, STY.
- Miscellaneous: LB, LBG, Message Security Assist, HFP multiple-add/subtract, and others.

Assembly of the new z990 instructions and the long displacement versions of the z/Architecture requires the OPTABLE(UNI) or OPTABLE(YOP) assembler options. OPTABLE(YOP) is available with HLASM APAR PQ74561 and release 2.2.17 of the Tachyon z/Assembler.

Long displacements can be explicitly specified or computed automatically from the assembler's USING statements. The default maximum USING range is 4K times the number of base registers specified on the USING, so you must specify an explicit USING range to get a large positive displacement. There is no minimum to the USING range, so negative displacements are always available. When multiple address resolutions are available, the smallest negative displacement is only used if a positive displacement cannot be calculated.

The assembler's USING(LIMIT(xxxx)) option is almost useless when used with long displacement instructions since the maximum LIMIT value is 4095.

z/OS 64-Bit Overview

Changes

- Real storage above 2 Gigabytes (as of OS/390 2.10).
Impacts programs doing page fixing, CCW translation.
Expanded storage is no longer available.
ESO Hiperspaces are not supported.
- Virtual storage above 2 Gigabytes (as of z/OS 1.2).
New IARV64 service to manage storage “above the bar”.
Data spaces are still supported, but are limited to
2 Gigabytes each.
- Binder and loader support for AMODE 64 (z/OS 1.3).
Both older Load Module Format (LMF) and new
Program Objects can be marked as AMODE 64.
- No support yet for PC routines entered in AMODE 64.

Except for programs performing CCW translation, very few older programs are impacted by running z/OS in z/Architecture mode. However, there are exceptions.

One example: The VS/PASCAL library routines issue TM 116,X'80' to check if AMODE 31 is available. Until z/Architecture, this was the high bit of the address word of the Machine Check New PSW. This bit was 0 for MVS/370 and 1 for MVS/XA and higher. In z/Architecture mode, location 116 is zero because the Machine Check New PSW is in a different location and format. So, on a z/Architecture system, VS/PASCAL programs think they are running on MVS/370 but will fail because the modules are marked RMODE ANY and loaded above the 16 Megabyte line. Lesson: Programs that access the PSA may not work in z/Architecture mode.

z/OS 64-Bit Overview

RMODE 64?

z/OS is a hybrid 24/31/64-bit operating system. All three addressing modes are supported and data maybe stored in any of the three address ranges. Executable instructions must be in 24-bit or 31-bit addressable storage, even if running in AMODE 64. RMODE 64 is not yet supported and may never be.

The TCB, RB, EPIE and SDWA control blocks have not been changed to allow the 128-bit PSW to be stored. This means that in order to store the PSW in 64-bit format, the instruction address must be below the 2 Gigabyte “bar” or else “bad things” will happen.

Interpreted programs (JAVA, REXX) could be loaded above 2 Gigabytes.

As long as the instruction address is below 2 Gigabytes, the first 33 bits and last 31 bits of the 128-bit PSW can be mapped into a 64-bit PSW with no loss of information. However, if the instruction address is above 2 Gigabytes, the address bits above 31 will be lost when the PSW is saved in the TCB or RB.

If a program executes instructions “above the bar”, it must be disabled for interrupts. Otherwise, the interrupt handlers may save the program's PSW in 64-bit format and when the PSW is restored, the program will restart in a different location. This is usually considered to be a “bad thing”.

z/OS 64-Bit Overview

AMODE 64!

Programs running in AMODE 64 can address all virtual storage. Programs running in AMODE 31 cannot access virtual storage above 2G.

Programs can switch to AMODE 64 from 24 or 31-bit mode using the BSM, BASSM or SAM64 instructions or can be started in AMODE 64.

When switching from 24 to 31-bit mode, programs had to ensure that the low 7 bits of the high-order bytes of the general registers were set to zero. When switching from 31 to 64-bit mode, the high-order 33 bits must be zeroed! This must be done using the new z/Architecture instructions (like LLGTR)

```

CODE31  CSECT
CODE31  AMODE 31
        BASR 12,0           Set a base register
        USING *,12
        LA 13,SAVEAREA     Get address of save area
        LG 15,=AD(CODE64+1) Get address of CODE64 + AMODE
        BASSM 14,15       Call CODE64 in AMODE 64
        SVC 3             Done
        LTORG
SAVEAREA DC 18F'0'
        DROP

CODE64  CSECT
CODE64  AMODE 64
        USING *,15        R15 points here
        STMH 14,12,SAVEHIGH+12 Save upper register halves
        LLGTR 13,13       Ensure R13 has a good address
        STM 14,12,12(13)   Save lower register halves
* do something interesting here
        LMD 14,12,SAVEHIGH+12,12(13) Restore register values
        LA 15,0           Set R15 to zero
        BSM 0,14         Return to caller
SAVEHIGH DC 18F'0'
        END CODE31

```

z/OS 64-Bit Overview

Data Spaces

For programs that are virtual-storage constrained, 64-bit virtual storage is an alternative to data spaces.

Data spaces are limited to 2 Gigabytes. At any time only 15 are addressable (using access registers 1-15), limiting the virtual storage data space to 30 Gigabytes. 64-bit virtual storage allows over 16,000,000,000 Gigabytes (if there were only enough disk space on Earth for paging.)

Data spaces still provide better storage isolation since a valid ALET is required to access them. It is easier for a cross-memory program to examine and modify another address space's 64-bit virtual storage area than for it to examine or modify data space storage.

Note: The 2 Gigabyte limit on the size of each Data Space is imposed by z/OS, not z/Architecture.

Fun facts:

Time needed to page-in an area at 100 Megabytes per second:

1 Megabyte (Segment)	.01 second
2 Gigabytes (Region 3 rd)	20 seconds
4 Terabytes (Region 2 nd)	12 hours
8 Petabytes (Region 1 st)	3 years
16 Exabytes (Address Space)	5845 years

z/OS 64-Bit Overview

IARV64

The IARV64 service is available to AMODE 31 and AMODE 64 programs to allocate virtual storage “above the bar”. Storage is allocated in 1 Megabyte blocks. The storage range between 2 Gigabytes and 4 Gigabytes is never allocated - the “bar” is 2 Gigabytes wide!

The MEMLIMIT JCL operand of the JOB or EXEC statement must be set to a non-zero value to allow IARV64 to function. MEMLIMIT can range from 0 to 99999 in units of Megabytes, Gigabytes, Terabytes or Petabytes or be set to NOLIMIT.

An allocation unit of 1 Megabyte might seem large but compared to the address space size, 1 Megabyte is much smaller than the 8 byte allocation unit for “below the bar” allocations. There is really no point to allocation of space “above the bar” unless a program uses very large amounts of virtual storage.

z/OS 64-Bit Overview

IARV64 Functions

IARV64 is the “one-stop shopping for all your *above-the-bar* virtual storage needs”.

IARV64 service

GETSTORE
DETACH
PAGEIN
PAGEOUT
PAGEFIX
PAGEUNFIX
DISCARDATA
CHANGEGUARD
LIST

31-bit service

GETMAIN/STORAGE OBTAIN
FREEMAIN/STORAGE RELEASE
PGSER LOAD
PGSER OUT
PGSER FIX
PGSER FREE
PGSER RELEASE
PGSER PROTECT (somewhat)
VSMLIST

GUARD AREA

When a block of virtual storage is allocated, a “guard area” may be allocated either just above or below the block of virtual storage. The guard area is not addressable but the virtual storage address range is reserved.

The addressable block may be expanded or reduced in size, reducing or expanding the size of the guard area. The guard area does not count against the MEMLIMIT, so you can allocate as large of a guard area as you want.

The guard area allows a virtual storage area to grow contiguously up to a preset limit without precommitting system resources. This can be very useful for stack-like (LIFO) storage areas.

z/OS 64-Bit Overview

Changed Services

Watch out for the following services when running in AMODE 64:

- Recovery: ESPIE, ESTAE
- Contents supervision: ATTACH, LINK, LOAD, SYNCH, XCTL.
- I/O: EXCP

Most other MVS services are unchanged when invoked in AMODE 64. They simply ignore the upper 32 bits of any parameter registers, assuming that the caller is in AMODE 31 and passing 31-bit addresses.

Use SYSSTATE to tell the z/OS macros which instructions to generate.

`SYSSTATE ASCENV={P|AR},AMODE64={NO|YES},ARCHLVL={0|1|2}`

With AMODE64=YES, system macros may get the CVT address using:

```
LLGT 14,16
```

instead of:

```
L 14,16
```

to ensure that bits 0-33 of the address are valid.

ARCHLVL values:

0: ESA/390

1: ESA/390 with the relative and immediate instructions required for OS/390 R10 (Architectural Level Set 1)

2: z/Architecture

With ARCHLVL set to 0 or 1, no z/Architecture-only instructions will be generated by the system macros.

z/OS 64-Bit Overview

Recovery Services

The EPIE control block passed to ESPIE exits and the SDWA control block passed to ESTAE exits has been expanded to allow the entire 64-bit general register values to be captured at the time of the error and to be changed upon retry. If the error occurred in AMODE 64, the exits need to be sensitive to the upper 32 bits of the general registers.

ESPIE and ESTAE exits are invoked in the AMODE at the time the ESPIE or ESTAE was established, not the AMODE at the time of the error.

Note: the 64-bit SDWA information is only available if SDWALOC31=YES was specified on the ESTAE macro.

Recovery may need to recognize new Program Check Interruptions, which are mapped to new S0C4 reason codes. All are caused by references to an invalid virtual storage address, just like Page Translation (S0C4-00000011) and Segment Translation (S0C4-00000010). The new codes are:

S0C4-00000038	ASCE Type
S0C4-00000039	Region First
S0C4-0000003A	Region Second
S0C4-0000003B	Region Third

EPIE Fields:

EPIEFLGS,EPIEUP64 - If set, use the 64-bit register values

EPIEG64 - 16 doubleword register values

Note: the 64-bit Translation Exception Address is in the SCA, not the EPIE.

SDWA fields:

SDWAXEME - Address of SDWARC4 section, containing the 16 64-bit general register values, the 16 64-bit control register values and the 64-bit Translation Exception Address. This section is only available if ESTAE SDWALOC31=YES was specified.

z/OS 64-Bit Overview

Contents Supervision

Programs in one AMODE can ATTACH, LINK, SYNCH or XCTL to a program with a different AMODE. Programs need to be careful when passing addresses to AMODE 64 programs since they will expect 64-bit clean addresses.

ATTACHX, LINKX, SYNCHX and XCTLX are supported in AMODE 64, not ATTACH, LINK, SYNCH and XCTL.

LINKX has two new parameters in z/OS 1.3: PLIST4|PLIST8 to specify 4-byte or 8-byte parameter addresses and AMODE64OK=YES|NO to indicate if an AMODE 31 program can LINK to an AMODE 64 program without an ABEND.

There are related changes for SYNCHX.

In z/OS 1.3, LOAD has a new parameter that is not directly related to AMODE 64. The EXTINFO parameter allows LOAD to return information about RMODE=SPLIT modules and modules larger than 128 Megabytes without the need for a separate call to the CSVQUERY service. The EXTINFO parameter area is mapped by the CSVEXTI macro.

z/OS 64-Bit Overview

I/O

I/O buffers and control blocks are not supported “above the bar”. Usually, buffers can be backed by real storage above 2 Gigabytes. The I/O services have been updated to build CCW IDAWs that address 64-bit real storage. The IDAWs themselves must still be below 2 Gigabytes because the CCW address field is still limited to 31 bits.

EXCPVR does not support real storage addresses above 2 Gigabytes. Buffers must be page fixed in 31-bit addressable real storage. Also, note that the LRA instruction is often used in EXCPVR code and will fail if the virtual storage is backed by real storage above 2 Gigabytes.

Both GETMAIN and STORAGE OBTAIN support the LOC=(,64) option to specify that the virtual storage can be backed by real storage above 2 Gigabytes. Do not specify LOC=(,64) for buffers intended to be used for EXCPVR.

The “old” LOC= names are mapped to new values:

BELOW => 24

ANY => 31

Use 24 and 31 instead of BELOW and ANY since 24 and 31 are more explicit in the 64-bit world.

AMODE 64 Linkage

ATTACH

Upon entry from ATTACH, the following registers are set for AMODE 64 programs:

R1: address of traditional parameter list in LOC=24 storage.
High-order 40 bits are zero.

R13: address of 144-byte save area in LOC=24 storage.
High-order 40 bits are zero.

R14: address of an SVC 3 instruction.
High-order 40 bits are zero.

R15: X'00000000 FFFFF002', indicating that the caller was in AMODE 64.

Upon entry to an AMODE 64 program, register 15 does not contain the entry address. An AMODE 64 program is expected to use the LARL instruction or another method to establish any needed base registers rather than relying on the value in R15.

The save area pointed to by R13 is 144 bytes and is big enough for a format-4 save area. However, C'F4SA' is not set at R13+4.

AMODE 64 Linkage

Save Areas

Format 0: traditional 72-byte save area containing only the low register halves.

Format 1: C'F1SA' in second word indicates that the registers are saved on the linkage stack.

Format 4: C'F4SA' in second word indicates a 144-byte save area containing the full 64-bit register values.

Format 5: C'F5SA' in second word indicates a 208-byte save area used when an AMODE 24 or 31 program calls an AMODE 64 program. The first 144 bytes are identical to the Format 4 save area. The upper halves of the caller's registers 0-15 are saved in the remaining 64 bytes.

System macro IHASAVER maps various save area formats.

Format 4 and 5 Save Area Contents

<u>Bytes</u>	<u>Contents</u>
0-3	Reserved for Language Environment
4-7	Value of C'F4SA' or C'F5SA'
8-15	64-bit register 14
16-23	64-bit register 15
24-127	64-bit registers 0-12
128-135	Address of previous save area (stored by calling program)
136-143	Address of next save area (stored by called program)
144-207	High halves of registers 0-15 (Format-5 only)

AMODE 64 Linkage

Using a Format-4 Save Area

Entry Linkage from AMODE 64 caller:

STMG	14,12,8(13)	Save caller's registers
LARL	14,SAVEAREA	Get new save area address
STG	13,128(,14)	Chain to previous save area
STG	14,136(,13)	Chain to new save area
LGR	13,14	Set R13 to save area address
LARL	12,DATA	Set base register for data
USING	DATA,12	Tell the assembler

Exit Linkage:

LGHI	15,0	Set return code
LG	13,128(,13)	Restore caller's R13
LG	14,8(,13)	Restore caller's R14
LMG	0,12,24(13)	Restore caller's R0-R12
BR	14	Return to caller

DATA	LTORG	Start of data area
SAVEAREA	DC	0D'0',F'0',C'F4SA',17FD'0'

This code is very similar to the standard linkage code typically used by AMODE 24 and AMODE 31 programs.

According to the *z/OS MVS Programming: Assembler Services Guide*:

Unless otherwise defined by the individual interface, the calling program should expect, upon return, that

- The low halves (Bits 32-63) of GPRs 2 through 13 are unchanged
- The high halves (Bits 0-31) of GPRs 2 through 14 are unchanged
- ARs 2 through 13 are unchanged
- FPRs 8 through 15 are unchanged; The Floating Point Control (FPC) Register is unchanged with the exception of two fields: the IEEE exception flags and the data exception code (DXC).
- When return information is provided in GPR 0, 1, and/or 15 (for example return and reason codes), only bits 32-63 of the register contain the returned value.

AMODE 64 Linkage

Using a Format-5 Save Area

Entry Linkage from AMODE 24/31 caller via BASSM:

STM	14, 12, 12 (13)	Save caller's registers
STMH	15, 15, 8 (13)	Save high half of R15
LARL	15, SAVEAREA	Get new save area address
STMH	0, 14, 144 (15)	Save upper register halves
MVC	204 (4, 15), 8 (13)	Save upper half of R15
LLGTR	13, 13	Ensure good address
STG	13, 128 (, 15)	Chain to previous save area
ST	15, 8 (, 13)	Chain to new save area
LGR	13, 15	Set R13 to save area address
LARL	12, DATA	Set base register for data
USING	DATA , 12	Tell the assembler
LLGTR	1, 1	Ensure R1 has a good address

Using a Format 5 Save Area for transition between AMODE 24/31 and AMODE 64 is much more complex than standard linkage. The upper register halves cannot be saved in one instruction because a 64-bit clean register is needed to point to the new save area.

Note: The Format 5 Save Area must be “*below the bar*” since there is only a fullword available in the standard 72-byte save area to point to the next save area.

AMODE 64 Linkage

Using a Format-5 Save Area

Exit Linkage back to AMODE 31/24 caller:

LHI	15, 0	Set return code
LG	13, 128 (, 13)	Restore caller's R13
ST	15, 16 (, 13)	Save return code
MVC	8 (4, 13), SAVEAREA+196	Save R13 upper half
LMH	0, 15, SAVEAREA+144	Restore upper halves
LLGTR	13, 13	Ensure R13 has good address
LM	14, 12, 12 (13)	Restore lower register halves
LMH	13, 13, 8 (13)	Restore upper half of R13
BSM	0, 14	Return to caller
DATA	LTORG	Start of data area
SAVEAREA	DC 0D'0', F'0', C'F5SA', 25FD'0'	

Exit code using the Format 5 Save Area is also much more complex than standard linkage. The code needs to be careful to avoid restoring the upper register halves of any registers it will use later.

Note: this code does not preserve bit 32 of register 13 because there is no defined place to save it in the Format 5 Save Area.

AMODE 64 Linkage

Using the Linkage Stack

Entry Linkage from caller via BASSM:

BAKR	14,0	Save caller's registers
LARL	13,SAVEAREA	Get dummy save area address
LARL	12,DATA	Set base register for data
USING	DATA,12	Tell the assembler

* The following code is needed if the caller might have been in AMODE 64:

LA	2,1	Get caller's
ESTA	2,2	PSW
TMLL	2,1	Caller was in AMODE 64?
JNZ	Entered	Yes: jump
LLGTR	1,1	Ensure R1 has a good address

Entered ds 0h

Exit Linkage back to caller:

LGHI	15,0	Set return code
PR	,	Return to caller
DATA	LTORG	Start of data area
SAVEAREA	DC	0D'0',F'0',C'F1SA'

Using the linkage stack is much simpler than using save areas. The BAKR and PR instructions are much slower than using STM and LM for saving registers, but they are much faster than using GETMAIN/FREEMAIN to obtain and release storage for a save area.

Also, unlike the previous linkage examples, this code make no assumptions about the AMODE of the caller. It works if the caller was in AMODE 24, 31 or 64. However, this code must be entered using BASSM even if the caller is also in AMODE 64 to ensure that R14 contains the AMODE bits. The BASR instruction does not set the low bit to indicate that the caller is AMODE 64.

This code fully preserves all 64 bits of registers R2-R14 and all 32 bits of access registers AR2-AR14. If the upper halves of R0, R1 or R15 or access registers AR0, AR1 or AR15 must be preserved, use the EREGG instruction before the PR.

Trimodal Code

Code that works in any AMODE

Example of code that works with 3 and 4-byte addresses in any AMODE:

	USING	PSA, 0	Addressability to PSA
	LA	R2, 0	Needed for AMODE 64
	L	R2, PSATOLD	Get current TCB address
	USING	TCB, R2	Addressability to TCB
	L	R2, TCBRBP	Get current RB address
	USING	RBBASIC, R2	Addressability to RB
LOOP	DS	0H	
	TM	RBSTAB2, RBTCBNXT	First RB found?
	BE	FOUND	yes, branch
	ICM	R2, 7, RBLINKB	Get next RB address
	B	LOOP	Check if first RB yet
FOUND	DS	0H	

This is an example of code that works in any of the three addressing modes. It is identical to code that might have been written for an MVS/XA AMODE 31 program except for the addition of the Load Address instruction and it would still work on that system.

In AMODE 24 and AMODE 31, the LA instruction would clear the low 32 bits of register 2 before loading the TCB address. This has no effect in AMODE 24 or AMODE 31 except for the very small overhead of the LA instruction.

In AMODE 64, the LA instruction clears all 64 bits of register 2 before the low 32 bits are loaded. This ensures that register 2 contains a valid 64-bit address.

An LA for address 0 should be used to clear a register before an ICM to get a 3-byte address. Traditionally an SR, SLR or XR instruction is used to clear the register but none of these clear the upper register half in AMODE 64. The LA instruction is just as fast so there is no reason not to use it instead of SR, SLR or XR.

With a little care, it is often not hard to write code that will work in any addressing mode.

Sample Program

```
Punch ' MODE RMODE(SPLIT) '
Print NoGen
SYSSTATE AMODE64=YES, ARCHLVL=2
* This OPSYN renames BAS to BRAS to keep OPEN and CLOSE happy
BAS      Opsyn BRAS

Entry    Csect                                Start of RMODE 24 area (data)
Entry    Amode 64
Entry    Rmode 24
STMG     R14, R12, 8(R13)  Save caller's registers
LARL     R14, SaveArea    Get save area address
MVC      4(4, R13), =C'F4SA' - SaveArea(R14)
STG      R13, 128(, R14)  Chain to previous save area
STG      R14, 136(, R13)  Chain to next save area
LGR      R13, R14        Get new save area address
Using    (SaveArea, EntryEnd), R13 Tell the assembler
MVC      SaveArea+4(4), =C'F4SA' Indicate format 4
LLGT     R15, =A(Main)   Get address of RMODE 31 area
BR       R15              and go to it
```

This example program reverses the records in a sequential data set. This data set can be quite large - it might be on tape. This program could handle a data set as large as 10 Gigabytes and can handle even larger data sets by changing one statement.

The program reserves a virtual storage address range of 10 Gigabytes and obtains an initial allocation of 1 Megabyte. The program then opens the data set and copies the records into the allocated space, increasing the allocation by 1 Megabyte at a time as needed. Once the records are read, it closes the data set, reopens it for output and writes the records in reverse order. All I/O is performed using QSAM in AMODE 31.

Initially and most of the time the program runs in AMODE 64, switching to AMODE 31 only to perform I/O operations. The program is split into RMODE 24 and RMODE 31 sections. The RMODE 24 section holds the I/O control blocks (the DCB must still be "below the line") and all modifiable storage. The RMODE 31 section contains the program logic and the static storage. This program requires z/OS 1.3 or later because the entry point is AMODE 64.

The entry point is in the RMODE 24 section. The initial code sets up the save area and branches to the code in the RMODE 31 section.

```

SaveArea Dc      18FD'0'
Origin   Dc      AD(0)           Starting address of file storage
High    Dc      AD(0)           Ending address of file storage
DcbIn   DCB     DDNAME=SYSUT1,DSORG=PS,MACRF=(GL),
          RECFM=FB,LRECL=1000,DCBE=DcbeIn
DcbeIn  DCBE    RMODE31=BUFF,EODAD=Eof
DcbOut  DCB     DDNAME=SYSUT1,DSORG=PS,MACRF=(PL),
          RECFM=FB,LRECL=1000,DCBE=DcbeOut
DcbeOut DCBE    RMODE31=BUFF
          IARV64 MF=(L,Iarv64) IARV64 parameter area

          Ltorg           Dump RMODE 24 literals here

EntryEnd ds      0d           End of RMODE 24 section

          YREGS

          Print Gen

Main    Rsect           Start of RMODE 31 area (program)
Main    Amode 64
Main    Rmode 31
          LARL R12,Static      Get static data area address
          Using (Static,MainEnd),R12 Tell the assembler
          SGR R2,R2           Clear record counter

```

The I/O control blocks and modifiable storage areas follow the entry logic in the RMODE 24 section. DCBEs are required to support QSAM buffers and the EODAD address “above the line”.

“Origin” will contain the starting address of the storage obtained to hold the records. “High” will contain the ending address (+1) of the area and will be incremented by 1 Megabyte as the area is expanded.

The RMODE 31 section is named “Main”. The code sets up a base register for the static storage area and then clears the record counter in R2.

```

IARV64 REQUEST=GETSTOR, Allocate 1 megabyte,
      SEGMENTS==AD(MaxSize), but reserve much
      GUARDLOC=HIGH, more in case the file
      GUARDSIZE==A(MaxSize-1), is very large
      ORIGIN=Origin,
      MF=(E,Iarv64,COMPLETE)
LG R4,Origin Get starting address
LGR R0,R4 Compute ending address
AGF R0,=A(1024*1024) *
STG R0,High *

SAM31 Switch to AMODE 31
OPEN (DcbIn,(INPUT)),MODE=31 Open the data set

GetLoop ds 0h
SAM31 Switch to AMODE 31
LA R1,DcbIn Get DCB address
GET (1) Read next record
SAM64 Switch to AMODE 64
LAY R0,RecordLength(,R4) Get address of next slot
CLG R0,High Room for this record?
JNH MoveIt yes: jump

```

The program reserves a 10 Gigabyte virtual storage address range “above the bar” using the IARV64 service. Initially, only the first Megabyte of the range is actually allocated. “Origin” and “High” are set to the starting and ending addresses of the allocated area.

The program switches to AMODE 31 to perform the OPEN service on the data set. It then starts reading the records using QSAM Locate Mode.

After each record is read, the program ensures that the allocated area is large enough to hold the next record.

```

LR      R3,R1          Save R1
IARV64 REQUEST=CHANGEGUARD, Allocate another
        CONVERT=FROMGUARD,   Megabyte of storage
        CONVERTSIZE==F'1',
        MEMOBJSTART=Origin,
        MF=(E,Iarv64,COMPLETE)
LR      R1,R3          Restore R1
LG      R0,High        Update the high storage address
AGF     R0,=A(1024*1024) to indicate it is
STG     R0,High        1 Megabyte larger
MoveIt  ds             0h
        LLGTR R0,R1          Convert to 64-bit address
        LGHI  R1,RecordLength Copy record from buffer
        LGHI  R5,RecordLength *
MoveIn  MVCLE R4,R0,0     *
        BRC   1,MoveIn      *
        AGHI  R2,1          Increment record count
        J     GetLoop       Go read next record

Eof     ds             0h          Go here upon End Of File
        CLOSE (DcbIn),MODE=31 Close the data set
        CGHI  R2,1          At least 2 records?
        JNH   Done          no: jump

```

If there is not enough room in the storage area for the next record, the program uses the IARV64 REQUEST=CHANGEGUARD service to allocate another Megabyte from the guard area. The program then updates the ending address in “High”.

Once there is enough room in the storage area for the record, the record is copied from the QSAM buffer into the next slot in the storage area and the record counter is incremented.

Upon End-Of-File, the data set is closed and the record counter is checked. If zero or one records were read, the program ends because there is nothing to reverse.

```

PutLoop  OPEN  (DcbOut, (OUTPUT)), MODE=31  Open data set
         ds    0h
         SAM31                               Switch to AMODE 31
         LA    R1, DcbOut                     Get DCB address
         PUT   (1)                            Get location of next record
         SAM64                               Switch to AMODE 64
         LLGTR R0, R1                         Convert to 64-bit address
         LAY   R4, -RecordLength(, R4)        Get address of record
         LGHI  R1, RecordLength              Copy record to buffer
         LGHI  R5, RecordLength              *
MoveOut  MVCLE R0, R4, 0                      *
         BRC   1, MoveOut                     *
         LAY   R4, -RecordLength(, R4)        Get address of record
         BRCTG R2, PutLoop                    Go write next record

         SAM31                               Switch to AMODE 31
         CLOSE (DcbOut), MODE=31            Close the data set

```

To reverse the records in the data set, the program next opens the data set for output. It then writes the records in reverse order using QSAM Locate Mode. Each record is copied from the storage area into the QSAM buffer. Once all records are written, the data set is closed.

```

Done      ds      0h
          SAM64          Switch to AMODE 64
          IARV64 REQUEST=DETACH, Free the file storage
          MEMOBJSTART=Origin,
          MF=(E,Iarv64,COMPLETE)

          LGHI R15,0      Set return code zero
          LG   R13,128(,R13) Restore caller's R13
          LG   R14,8(,R13) Restore caller's R14
          LMG  R0,R12,24(R13) Restore caller's R0-R12
          BR   R14        Return to caller

Static    Ltorg          Dump RMODE 31 literals here

MainEnd   ds      0d          End of RMODE 31 section

RecordLength Equ 1000      Record size
MaxSize    Equ 10*1024     Maximum file size in Megabytes

          End    Entry

```

When the program is done, it releases the allocated virtual storage and the reserved address range using IARV64. It then sets the return code to zero and returns to the caller.

The data set is arbitrarily defined as containing 1000-byte fixed length records for this example. It could easily be changed to use the LRECL from the DCB instead of a constant. Also, "MaxSize" can be easily changed to be larger if needed.

The MEMLIMIT parameter of the EXEC JCL statement must be set to a value large enough to hold the entire data set in virtual storage. This can impact the size of page data sets if there is not enough real memory to run this program.

Need More Information?

z/OS 1.4 MVS Programming: Assembler Services Guide
SA22-7605-04

z/OS 1.4 MVS Programming: Assembler Services Reference,
Volume 1 SA22-7606-03

z/OS 1.4 MVS Programming: Assembler Services Reference,
Volume 2 SA22-7607-03

z/OS 1.3-1.4 MVS Programming: Extended Addressability Guide
SA22-7614-02

z/Architecture Principles of Operation
SA22-7832-02

Chapter 4 of the *z/OS 1.3 and 1.4 MVS Programming: Extended Addressability Guide* is entitled “Using the 64-bit Address Space”. This chapter is probably the most readable description of how and why to use 64-bit assembler code.