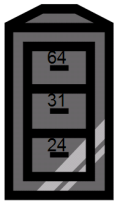


The 64-bit address space

Because of changes in the architecture that supports the Multiple Virtual Storage (MVS) operating system, there have been two different address spaces prior to the 64-bit address space. The address space of the 1970s began at address 0 and ended at 16 megabytes. The architecture that created this address space provided 24-bit addresses.

In the early 1980s, XA (extended architecture) introduced an address space that began at address 0 and ended at two gigabytes. The architecture that created this address space provided 31-bit addresses. To maintain program compatibility, MVS provided two *addressing modes* (AMODEs):

- Programs that run in AMODE 24 can use only the first 16 megabytes of the address space
- Programs that run in AMODE 31 can use the entire 2 gigabytes.

As of z/OS v1.2, the address space begins at address 0 and ends at 16 exabytes, an incomprehensibly high address. The architecture that creates this address space provides 64-bit addresses. The address space structure below the 2 gigabyte address has not changed; all programs in AMODE 24 and AMODE 31 continue to run without change. In some fundamental ways, the address space is much the same as the XA address space.

In the 31-bit address space, a virtual **line** marks the 16-megabyte address. The 64-bit address space also includes the virtual line at the 16-megabyte address; additionally, it includes a second virtual line called **the bar** that marks the 2-gigabyte address. The bar separates storage below the 2-gigabyte address, called **below the bar**, from storage above the 2-gigabyte address, called **above the bar**. The area above the bar is intended for data; no programs run above the bar. There is no area above the bar that is common to all address spaces, and no system control blocks exist above the bar. IBM reserves an area of storage above the bar for special uses to be developed in the future.

You can set a limit on how much virtual storage above the bar each address space can use. This limit is called the **MEMLIMIT**. If you do not set a MEMLIMIT, the system default is 2G, meaning that the address space can use up to 2G of virtual storage above the bar. If you want an address space to have access to more or less virtual storage above the bar, you need to explicitly set the MEMLIMIT to the limit you want. You can set an installation default MEMLIMIT through System Management Facility (SMF). You can also set a MEMLIMIT for a specific address space in the job control language (JCL) that creates the address space or by using SMF exit IEFUSI.

Note: IEFUSI receives control before each job step is started (prior to allocation). A return code from this exit indicates whether the job step is to be started or the job should be cancelled.

You can use IEFUSI to:

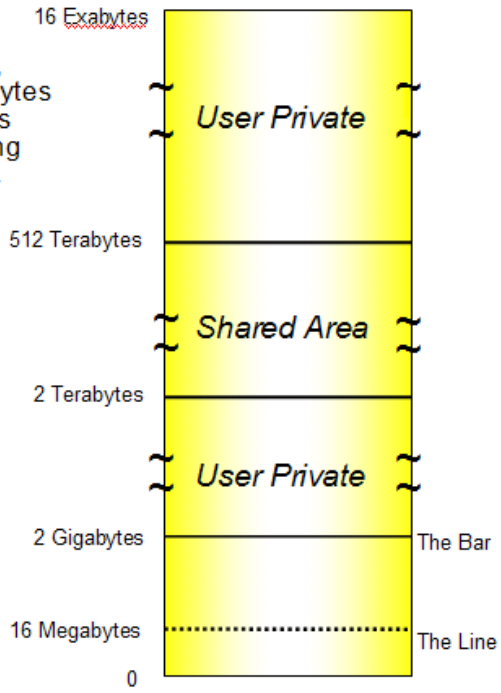
- Validate job step accounting information.
- Write to a user data set.
- For long-running jobs, create and write a user step-initiation SMF record in case of system failure.
- Set the region size and region limit for all programs that run under this job step.
- Set limits on the use of data spaces and hiperspaces created by application programs with storage key 8-F.
- Limit the number of pages that can be shared at one time through the use of the IARVSERV macro.
- Set the default size of data spaces and hiperspaces.
- Limit the use of the 16 exabyte address space above two gigabytes.

For more information about controlling region size and region limit, see *z/OS' MVS Initialization and Tuning Guide*.

This figure displays a z/OS address space, including the line that marks the 16-megabytes address, the bar that marks the 2-gigabytes address and the default shared area starting at 2 terabytes and ending at 512 terabytes.

Reference:
1024 Kilobytes = 1 Megabyte
1024 Megabytes = 1 Gigabyte
1024 Gigabytes = 1 Terabyte
1024 Terabytes = 1 Petabyte
1024 Petabytes = 1 Exabyte

Future reference:
1024 Exabytes = 1 Zettabyte
1024 Zettabyes = 1 Yottabyte
1024 Yottabytes = 1 Brontobyte
1024 Brontobytes = 1 Geopbyte



Before z/OS v1.3, all programs in AMODE 31 or AMODE 24 were unable to work with data above the bar. To use virtual storage above the bar, a program must request storage above the bar, be in AMODE 64, and use new z/Architecture assembler instructions.

As of z/OS v1.5, the following enhancements for 64-bit virtual storage have been added:

- 64-bit shared memory support
- Multiple guard area support for private high virtual storage
- Default shared memory addressing area between 2 terabytes and 512 terabytes.

Why would you use virtual storage above the bar?

The reason why someone designing an application would want to use the area above the bar is simple: the program needs more virtual storage than the first 2-gigabyte address space provides. Before z/OS v1.2, a program's need for storage beyond what the former 2-gigabyte address space provided was sometimes met by creating one or more data spaces or hiperspaces and then designing a memory management schema to keep track of the data in those spaces.

Sometimes programs written before v1.2 used complex algorithms to manage storage, reallocate and reuse areas, and check storage availability. With the 16-exabyte address space, these kinds of programming complexities are unnecessary. A program can potentially have as much virtual storage as it needs, while containing the data within the program's primary or home address space.

A good example of a programming model that can successfully take advantage of the 16-exabyte address space is a program that needs very large buffer pools. This program has typically used multiple data spaces and then managed them separately and uniquely.

With the 16-exabyte address space, a program can use the area above two gigabytes for a buffer pool. A simple memory mapping scheme is all that is needed to keep track of the data.

Memory management above the bar

Virtual memory above 2GB is organized as memory objects that a program creates. A memory object is a contiguous range of virtual addresses that are allocated by programs as a number of application pages which are 1MB multiples on a 1MB boundary. Programs continue to run and execute in the first 2GB of the address space.

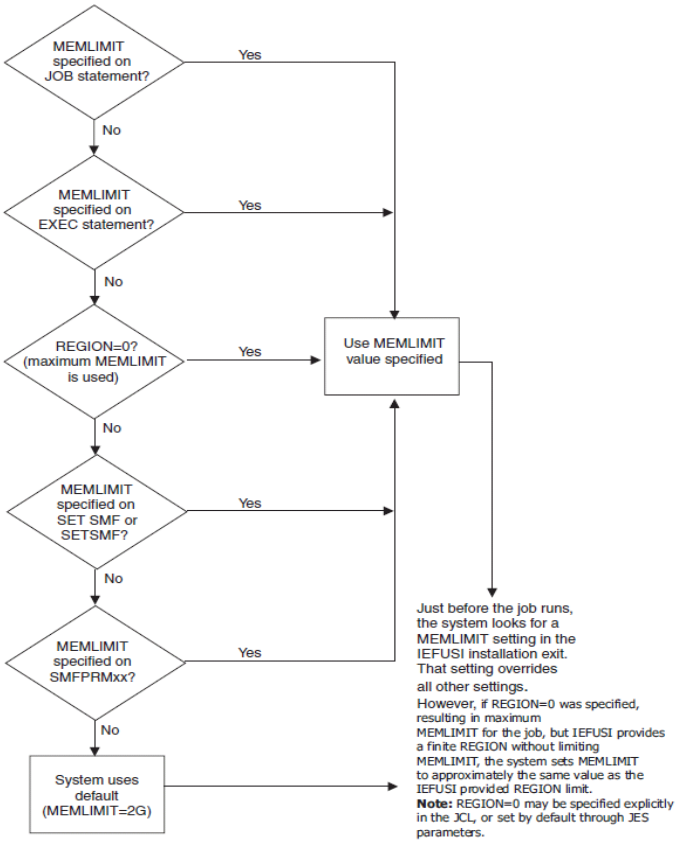
Limiting the use of memory objects

While there is no practical limit to the virtual storage above the bar, practical limits exist to the real storage frames and auxiliary storage slots that back the area. To control the amount of real and auxiliary storage that an address space can use for memory objects at one time, your installation can establish an installation default MEMLIMIT that sets the total number of usable virtual pages above the bar for a single address space.

You set this default in two ways:

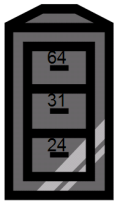
- On the MEMLIMIT parameter in the SMFPRMxx *parmlib* member
- Through the SET SMF or SETSMF console commands.

The default takes effect if a job does not specify MEMLIMIT on the JCL JOB or EXEC statement, or REGION=0 in the JCL; the MEMLIMIT specified in an IEFUSI exit routine overrides all other MEMLIMIT settings. If REGION=0 is specified in the JCL and the IEFUSI exit limits the REGION size but does not set MEMLIMIT, MEMLIMIT is defaulted to the REGION size above 16MB.



This flow chart illustrates how the system chooses which MEMLIMIT applies

The system enforces the MEMLIMIT when you issue the IARV64 GETSTOR and CHANGEGUARD macro services. When your unconditional request for new storage (either for



a new memory object or for more usable storage in an existing memory object) causes the MEMLIMIT to be exceeded, the system abends the program. IBM recommends that programs use the COND parameter to make a conditional request and check the return code to make sure the storage is available.

If a SET SMF or SETSMF console command changes the default MEMLIMIT (either the system default or the installation default) for an address space that is already created the following changes occur:

- If the command raises the current default MEMLIMIT, all address spaces whose MEMLIMIT values are set through SMF run with the higher default.
- If the command lowers the current default MEMLIMIT, all address spaces whose MEMLIMIT values are set through SMF keep their original (higher) system default.

Note: You cannot use SET SMF or SETSMF console command to change the MEMLIMIT value that is set through JCL.

Historically, users could limit program storage below 16 megabytes in virtual storage by using IEALIMIT. IEALIMIT can still be used to limit program storage in the nonextended region; however, IEFUSI is the preferred exit routine

Memory objects

Programs obtain storage above the bar in "chunks" of virtual storage called **memory objects**. The system allocates a memory object as a number of virtual segments; each segment is a megabyte in size and begins on a megabyte boundary. A memory object can be as large as the memory limits set by your installation and as small as one megabyte. Other attributes of a memory object include the following characteristics:

- The storage key‡ is defined by the program; for an unauthorized program, the storage key at the time of issuing the IARV64 macro is the program's PSW key.
- You can specify whether you want the memory object to be fetch protected or not. There is no change key support for virtual storage above the bar.
- The owner of a private memory object is the TCB of the program that creates the private memory object, or a TCB to which the creating program assigns ownership. If an SRB creates a private memory object, the SRB must assign ownership of the private memory object to a task.
- A shared memory object is system owned. The cross-memory resource owner (CMRO) TCB of the address space owns the shared interest in the shared memory object.

Using the IARV64 macro, a program can create and free a memory object and manage the physical frames that back the virtual storage. You can think of IARV64 as the GETMAIN /FREEMAIN or STORAGE macro for virtual storage above the bar. (GETMAIN/FREEMAIN and STORAGE do not work on virtual storage above the bar.)

When a program creates a memory object, it provides an area in which the system returns the memory object's low address. You can think of the address as the name of the memory object. After creating the memory object, the program can use the storage in the memory object as it used storage in the 2-gigabyte address space.

Relationship between the memory object and its owner

When your program creates shared memory objects you need to understand ownership issues in order to prevent illegal operations that will be identified by an ABEND from z/OS. A program creates a shared memory object, but it does not own the shared memory object. A shared

‡ The storage key – A means for z/OS to provide runtime integrity when a request is made to modify the contents of a central storage location; the key associated with the request is compared to the storage protect key. If the key associated with the request does not match the storage key, the system rejects the request and issues a program exception or abend.

memory object is always owned by the system. A program gains access to a shared memory object by creating an interest in the shared memory object. Shared interest is owned by the CMRO (cross memory resource owner) TCB of an address space. Once a program has gained access to a shared memory object, any program running in that address space has access to the shared memory object.

If the unit of work is an SRB, the program must assign ownership to a TCB. Because of this assignment of ownership, the owner of the memory object and the creator of the memory object might not always be the same.

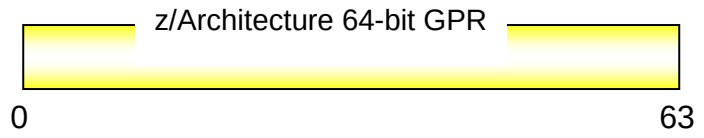
Using assembler instructions in the 64-bit address space

With z/Architecture, two facts are prominent: the address space is 16 exabytes in size, and the General Purpose Registers (GPRs) are 64 bits in length. You can ignore these facts and continue to use storage below the bar. If, however, you want to enhance old programs or design new ones to use the virtual storage above the bar, you will need to use the new Assembler instructions.

z/Architecture provides two new major capabilities that are related but are also somewhat independent:

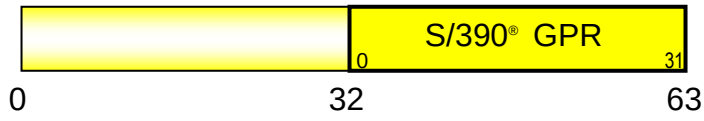
- 64-bit binary operations
- 64-bit addressing mode (AMODE).

64-bit binary operations perform arithmetic and logical operations on 64-bit binary values. 64-bit AMODE allows access to storage operands that reside anywhere in the 16-exabyte address space. In support of both, z/Architecture extends the GPRs to 64 bits. There is a single set of 16 64-bit GPRs, and the bits in each are numbered from 0 to 63.

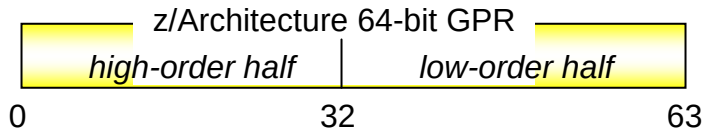


All S/390® instructions are carried forward into z/Architecture and continue to operate using the low-order half of the z/Architecture 64-bit GPRs. That is, an S/390 instruction that operates on bit positions 0 through 31 of a 32-bit GPR in S/390 operates instead on bit positions 32 through 63 of a 64-bit GPR in z/Architecture.

You can think of the S/390® 32-bit GPRs as being imbedded in the 64-bit GPRs.



Throughout the discussion of GPRs, bits 0 through 31 of the 64-bit GPR are called the **high-order half**, and bits 32 through 63 are called the **low-order half**.



64-bit addressing mode (AMODE)

When generating addresses, the processor performs address arithmetic; it adds three components: the contents of the 64-bit GPR, the displacement (a 12-bit value), and (optionally) the contents of the 64-bit index register. Then, the processor checks the addressing mode and

truncates the answer accordingly. For AMODE 24, the processor truncates bits 0 through 39; for AMODE 31, the processor truncates bits 0 through 32; for AMODE 64, no truncation (or truncation of 0 bits) occurs. In S/390 architecture, the processor added together the contents of a 32-bit GPR, the displacement, and (optionally) the contents of a 32-bit index register. It then checked to see if the addressing mode was 31 or 24 bits, and truncated accordingly. AMODE 24 caused truncation of 8 bits. AMODE 31 caused a truncation of bit 0.

The addressing mode also determines where the storage operands can reside. The storage operands for programs running in AMODE 64 can be anywhere in the 16-exabyte address space, while a program running in AMODE 24 can use only storage operands that reside in the first 16 megabytes of the 16-exabyte address space.

- - -