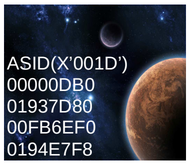


Virtual storage is an illusion created by z/Architecture together with z/OS, in that the programs seems to have more central storage that it really has. The virtual storage concept was introduced in the Atlas System used for the first time in the IBM System/370 architecture.
It is based on:

- An address (called virtual) as referred by a program is an identifier of a required piece of information in central storage. This allows the size of an address space (all virtual addresses available to a program) to exceed the central storage size.
- All central storage references are made in terms of virtual storage address.
- A hardware mechanism, dynamic addresss translation (DAT) is employed to perform a mapping between the virtual storage address and its physical location in central storage.
- When a requested address in not in central storage (we have more virtual addresses than bytes in central storage), an interruption is signaled and the required data is brought into memory.

In 1970, IBM introduced System/370, the first of its architectures to use virtual storage and address spaces. Since that time, the operating system has changed in many ways. One key area of growth and change is addressability.

A program running in an address space can reference all of the storage associated with that address space. In this text, a program's ability to reference all of the storage associated with an address space is called *addressability*.

System/370 defined storage addresses as 24 bits in length, which meant that the highest accessible address was 16,777,215 bytes (or 2 $^{24-1}$ bytes). The use of 24-bit addressability allowed MVS/370, the operating system at that time, to allot to each user an address space of 16 megabytes. Over the years, as MVS/370 gained more functions and was asked to handle more complex applications, even access to 16 megabytes of virtual storage fell short of user needs.

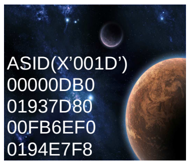
With the release of the System/370-XA architecture in 1983, IBM extended the addressability of the architecture to 31 bits. With 31-bit addressing, the operating system (now called MVS Extended Architecture or MVS/XA) increased the addressability of virtual storage from 16 MB to 2 gigabytes (2 GB). In other words, MVS/XA provided an address space for users that was 128 times larger than the address space provided by MVS/370. The 16 MB address became the dividing point between the two architectures and is commonly called the ***line***.
The new architecture did not require customers to change existing application programs. To maintain compatibility for existing programs, MVS/XA remained compatible for programs originally designed to run with 24-bit addressing on MVS/370, while allowing application developers to write new programs to exploit the 31-bit technology.

To preserve compatibility between the different addressing schemes, MVS/XA did not use the *high-order bit* of the address (Bit 0) for addressing. Instead, MVS/XA reserved this bit to indicate how many bits would be used to resolve an address: 31-bit addressing (Bit 0 on) or 24-bit addressing (Bit 0 off).

With the release of zSeries mainframes in year 2000, IBM further extended the addressability of the architecture to 64 bits. With 64-bit addressing, the potential size of a z/OS address space expands to a size so vast we need new terms to describe it. Each address space, called a 64-bit address space, is 16 exabytes (EB) in size; an exabyte is slightly more than one billion gigabytes. The 2GB address became another dividing point between the two architectures and is commonly called the ***bar***.

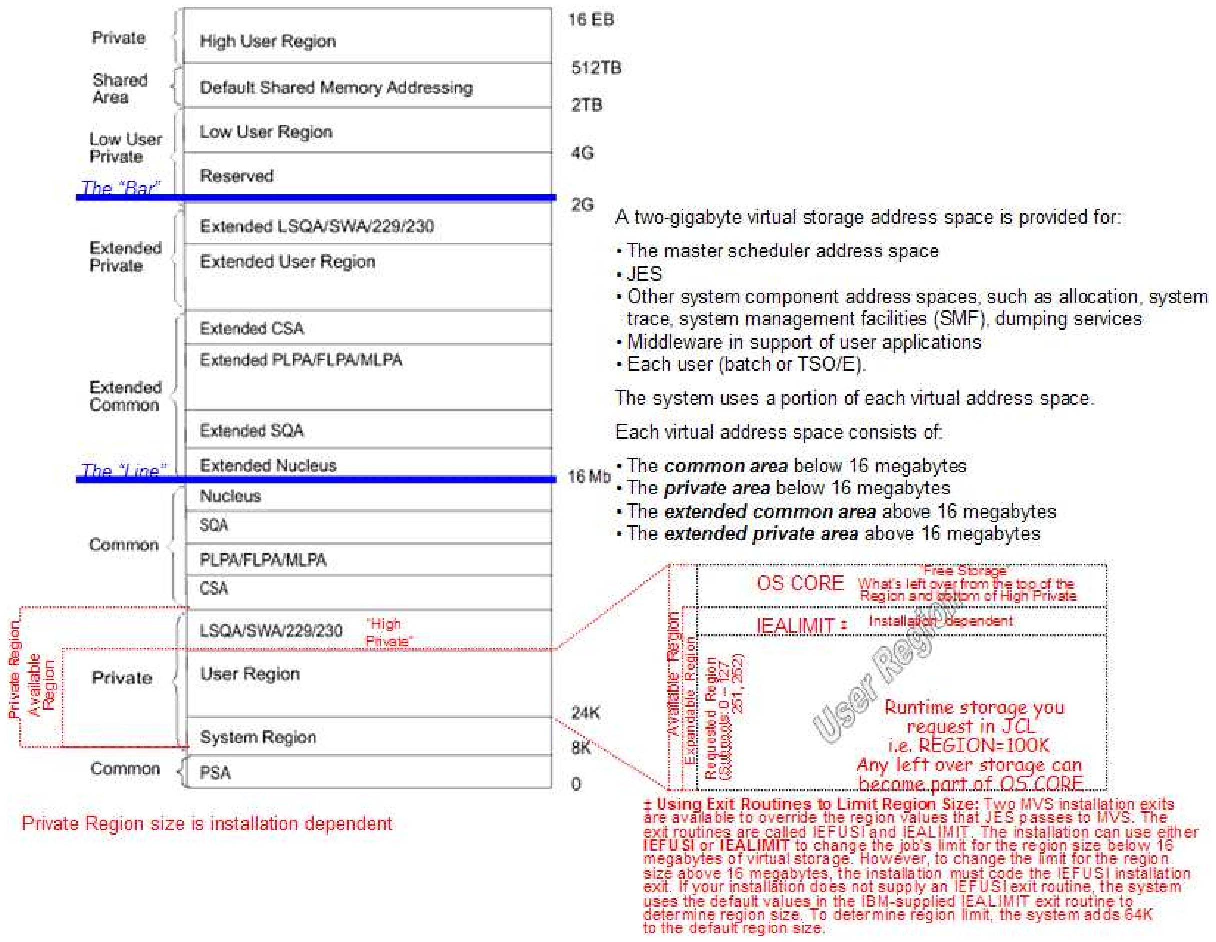
The new address space has logically 264 addresses. It is 8 billion times the size of the former 2-gigabyte address space, or 18,446,744,073,709,600,000 bytes.

We say that the potential size is 16 exabytes because z/OS, by default, continues to create address spaces with a size of 2 gigabytes. The address space exceeds this limit only if a program running in it allocates virtual storage above the 2-gigabyte address. If so, z/OS increases the storage available to the user from two gigabytes to 16 exabytes.



For compatibility, the layout of the storage areas for an address space is the same below 2 gigabytes, providing an environment that can support both 24-bit and 31-bit addressing.

Each storage area in the common area (below 16 M) has a counterpart in the extended common area (above 16 M), except the PSA. The CSA and SQA sizes are settled during the IPL, according to system initialization parameters in the SYS1.PARMLIB (IEASYSxx) system data set.



A two-gigabyte virtual storage address space is provided for:

- The master scheduler address space
- JES
- Other system component address spaces, such as allocation, system trace, system management facilities (SMF), dumping services
- Middleware in support of user applications
- Each user (batch or TSO/E).

The system uses a portion of each virtual address space.

Each virtual address space consists of:

- The ***common area*** below 16 megabytes
- The ***private area*** below 16 megabytes
- The ***extended common area*** above 16 megabytes
- The ***extended private area*** above 16 megabytes

± Using Exit Routines to Limit Region Size: Two MVS installation exits are available to override the region values that JES passes to MVS. The exit routines are called IEFUSI and IEALIMIT. The installation can use either **IEFUSI** or **IEALIMIT** to change the job's limit for the region size below 16 megabytes of virtual storage. However, to change the limit for the region size above 16 megabytes, the installation must code the IEFUSI installation exit. If your installation does not supply an IEFUSI exit routine, the system uses the default values in the IBM-supplied IEALIMIT exit routine to determine region size. To determine region limit, the system adds 64K to the default region size.

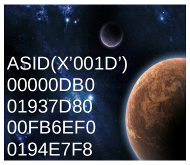
A way of thinking of an address space is as a programmer's map of the virtual storage available for code and data. Because it maps all of the available addresses, however, an address space includes system code and data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data.

In many problem determination situations you will need to understand which storage areas are affected and how to adjust those areas to avoid further abends. Let's explore these storage areas.

Note: IBM recommends common storage tracking always be activated. You can turn storage tracking on by activating a DIAGxx parmlib member, either at IPL time (specify DIAG=xx as a system parameter) or during normal processing (enter a SET DIAG=xx command).

STORAGE AREAS

Storage areas are described next beginning from the bottom up to the Nucleus. Each area except PSA has a counter part above the 16 megabyte line.



Prefixed Storage Area (PSA) – 8K in z/Architecture – Each processor in a system (or an LPAR) has a small private area of memory (8 KB starting at real address 0 and always mapped to virtual address 0) that is unique to each PU. This is the Prefix Storage Area (PSA) and is used for instruction execution such as PSW (old/new), interrupts, and error handling. A processor can access another processor's PSA through special programming, although this is normally done only for error recovery purposes. There is one PSA per processor and all map to virtual storage location 0-8192 (8K). z/Architecture treats this as a separate area. The function of the PSA is to map fixed hardware and software storage locations for the related server. The PSA does not have a counter part above the 16 megabyte line.

System Region - This area is reserved for GETMAINs by all system functions (for example, error recovery procedures - ERPs) running under the region control tasks. It comprises 16K (except in the master scheduler address space, in which it has a 200K maximum) of each private area immediately above the PSA. V=V region space allocated to user jobs is allocated upwards from the top of this area. This area is pageable and exists for the life of each address space.

The Private Area User Region/Extended Private Area User Region The portion of the user's private area within each virtual address space that is available to the user's problem programs is called the **user region.**

Types of User Regions - There are two types of user regions: virtual (or **V=V**) and real (or **V=R**). Virtual and real regions are mutually exclusive; private areas can be assigned to V=R or V=V, but not to both. It is the installation's responsibility to determine the type of region to which jobs are assigned. Usually, V=R should be assigned to regions containing jobs that cannot run in the V=V environment, or that are not readily adaptable to it. Programs that require a one-to-one mapping from virtual to central storage, such as program control interruption (PCI) driven channel programs, are candidates for real regions.
Two significant differences between virtual and real regions are:

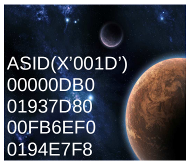
1. How they affect an installation's central storage requirements
2. How their virtual storage addresses relate to their central storage addresses.

For virtual regions, which are pageable and swappable, the system allocates only as many central storage frames as are needed to store the paged-in portion of the job (plus its LSQA- *see later*). The processor translates the virtual addresses of programs running in virtual regions to locate their central storage equivalent. For real regions, which are nonpageable and nonswappable, the system allocates and fixes as many central storage frames as are needed to contain the entire user region.

The virtual addresses for real regions map one-for-one with central storage addresses.

Virtual Regions: Virtual regions begin at the top of the system region and are allocated upward through the user region to the bottom of the area containing the LSQA, SWA, and the user key area (subpools 229, 230, and 249) – *see later*. Virtual regions are allocated above 16 megabytes also, beginning at the top of the extended CSA, and upward to the bottom of the extended LSQA, SWA, and the user key area (subpools 229, 230, and 249). As a portion of any V=V job is paged in, 4K multiples (each 4K multiple being one page) of central storage are allocated from the available central storage. Central storage is dynamically assigned and freed on a demand basis as the job executes. V=V region requests that specify a specific region start address, are supported only for restart requests, and must specify an explicit REGION size through JCL.

Specifying Region Size: Users can specify a job's region size by coding the REGION parameter on the JOB or EXEC statement. The system rounds all region sizes to a 4K multiple. The region size value should be less than the region limit value to protect against programs that issue variable length GETMAINs with very large maximums, and then do not immediately free part of that space or free such a small amount that a later GETMAIN (possibly issued by a system service) causes the job to fail. For V=V jobs, the region size can be as large as the entire private area, minus the size of LSQA/SWA/user key area (subpools 229, 230, and 249) and the system



region. For V=R jobs, the REGION parameter value cannot be greater than the value of the REAL system parameter specified at IPL. If the user does not explicitly specify a V=R job's region size in the job's JCL, the system uses the VRREGN system parameter value in the IEASYS00 member of SYS1.PARMLIB. For more information see JCL Reference manual.
Note: VRREGN should not be confused with the REAL system parameter. REAL specifies the total amount of central storage that is to be reserved for running all active V=R regions. VRREGN specifies the default subset of that total space that is required for an individual job that does not have a region size specified in its JCL. An installation can override the VRREGN default value in IEASYS00 by:

- Using an alternate system parameter list (IEASYSxx) that contains the desired VRREGN parameter value.
- Specifying the desired VRREGN value at the operator's console during system initialization. This value overrides the value for VRREGN that was specified in IEASYS00 or IEASYSxx. For V=R requests, if contiguous storage of at least the size of the REGION parameter or the system default is not available in virtual or central storage, a request for a V=R region is placed on a wait queue until space becomes available.

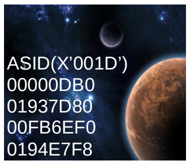
Note: V=R regions must be mapped one for one into central storage. Therefore, they do not have their entire virtual storage private area at their disposal; they can use only that portion of their private area having addresses that correspond to the contiguous central storage area assigned to their region, and to LSQA, SWA, and subpools 229, 230, and 249.

Real Regions: Real regions begin at the top of the system region and are allocated upward to the bottom of the area containing LSQA, SWA, and the user key area (subpools 229, 230, and 249). Unlike virtual regions, real regions are only allocated below 16 megabytes. The system assigns real regions to a virtual space within the private area that maps one-for-one with the real addresses in central storage below 16 megabytes. Central storage for the entire region is allocated and fixed when the region is first created.

Local System Queue Area (LSQA/Extended LSQA) Each virtual address space has an LSQA. The area contains TCBs, RBs, CDEs, SPQEs, DQEs, FQEs, tables and queues associated with the user's address space. LSQA is intermixed with SWA and subpools 229, 230, and 249 downward from the bottom of the CSA into the unallocated portion of the private area, as needed. Extended LSQA is intermixed with SWA and subpools 229, 230, and 249 downward from 2 gigabytes into the unallocated portion of the extended private area, as needed. LSQA will not be taken from space below the top of the highest storage currently allocated to the private area user region. Any job will abnormally terminate unless there is enough space for allocating LSQA.

Scheduler Work Area (SWA/Extended SWA) This area contains control blocks that exist from task initiation to task termination (subpools 236-237). It includes control blocks and tables created during JCL interpretation (JOBSTEP) and used by the initiator during job step scheduling. Each initiator has its own SWA within the user's private area. To enable recovery, the SWA can be recorded on direct access storage in the JOB JOURNAL. SWA is allocated at the top of each private area intermixed with LSQA and subpools 229, 230, and 249.

Subpools 229, 230, 249 - Extended 229, 230, 249 This area allows local storage within a virtual address space to be obtained in the requestor's storage protect key. The area is used for control blocks that can be obtained only by authorized programs having appropriate storage protect keys. These control blocks were placed in storage by system components on behalf of the user. These subpools are intermixed with LSQA and SWA. Subpool 229 is fetch protected; subpools 230 and 249 are not. All three subpools are pageable. Subpools 229 and 230 are freed automatically at task termination; subpool 249 is freed automatically at jobstep task termination.



Common Service Area (CSA and Extended CSA) The ***common area*** contains system control programs and control blocks. Here is where storage can be shared across address spaces within the logical partition (see #58 z/OS' Cross Memory). The following storage areas are located in the common area:

The common service area is a getmainable common area containing control blocks used by subsystem programs such as JES2, DFSMS, and RACF, and access methods like VSAM. It is a sort of virtual storage reserved area for future getmains/freemains issued by such programs. This area is used for buffers for IMS, ACF-VTAM, JES. It is also used for TSO buffer sharing (TIOC), Event Notification Facility (ENF) and Message Processing Facility (MPF).

CSA/ECSA normally contains data referenced by a number of system address spaces, enabling address spaces to communicate by referencing the same piece of CSA data. In a sense, CSA/ECSA looks like SQA/ESQA (*see SQA later*).

CSA is allocated directly below the MLPA (*see MLPA later*). ECSA is allocated directly above the extended MLPA. If the virtual SQA/ESQA space is full, z/OS allocates additional SQA/ESQA space from the CSA/ECSA. The size of the CSA/ECSA can be specified through the CSA parameter in the IEASYSxx member of SYS1.PARMLIB, or like any IEASYSxx parameter, through the operator in a z/OS console at IPL. The common service area (CSA) contains pageable and fixed data areas that are addressable by all active virtual storage address spaces.

Link pack area (LPA and Extended LPA) The link pack area contains programs that are preloaded at IPL time in the common area, from the SYS1.LPALIB data set. These programs can be certain z/OS SVC routines, access methods code, other read-only z/OS programs (the ones not modified along its execution), and any read-only reenterable user programs selected by an installation.

Because such code is in the common area, all these single copy programs can be executed in any address space. Their copy is not self-modifying (reentrant), meaning that the same copy of the module can be used by any number of tasks in any number of address spaces at the same time. This reduces the demand for central storage and lowers the program fetch overhead.

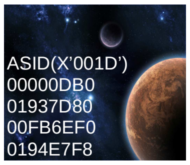
The LPA/ELPA size depends on the number of modules loaded in it. When modules are added to LPA, the growth in LPA can cause the common area to cross one or more segment (1 M) boundaries. This reduces the available private area for all address spaces, even for those address spaces not using the load modules added to LPA.

All modules placed in LPA are assumed to be APF-authorized. Being APF-authorized means that a program can invoke any SVC routine which accesses protected system and private areas. Although LPA boundaries cannot be changed after IPL, it is possible to dynamically include new load modules to LPA without an IPL. In this case, z/OS issues a Getmain from CSA/ECSA and uses such virtual storage area to load the load module.

The three areas of LPA are:

1. **Pageable link pack area (PLPA) -** This area contains SVC routines, access methods, and other read-only system programs along with any read-only reenterable user programs selected by an installation that can be shared among users of the system. Any module in the pageable link pack area will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected SYS1.LPALIB and any library named in LPALSTxx or on an LPA statement in PROGxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library.

   Modules loaded into the PLPA are packed within page boundaries. Modules larger than 4K begin on a page boundary with smaller modules filling out. PLPA can be used more efficiently through use of the LPA packing list (IEAPAKxx). IEAPAKxx allows an installation to pack groups of related modules together, which can sharply reduce page faults. The total



size of modules within a group should not exceed 4K, and the residence mode (RMODE) of the modules in a group should be the same. (*see RMODE later*).

2. **The Fixed Link Pack Area (FLPA)** An installation can elect to have some modules that are normally loaded in the pageable link pack area (PLPA) loaded into the fixed link pack area (FLPA). This area should be used only for modules that significantly increase performance when they are fixed rather than pageable. Modules placed in the FLPA must be reentrant and refreshable. The FLPA exists only for the duration of an IPL. Therefore, if an FLPA is desired, the modules in the FLPA must be specified for each IPL (including quick-start and warm-start IPLs). It is the responsibility of the installation to determine which modules, if any, to place in the FLPA. Note that if a module is heavily used and is in the PLPA, the system's paging algorithms will tend to keep that module in central storage. The best candidates for the FLPA are modules that are infrequently used but are needed for fast response to some terminal-oriented action.

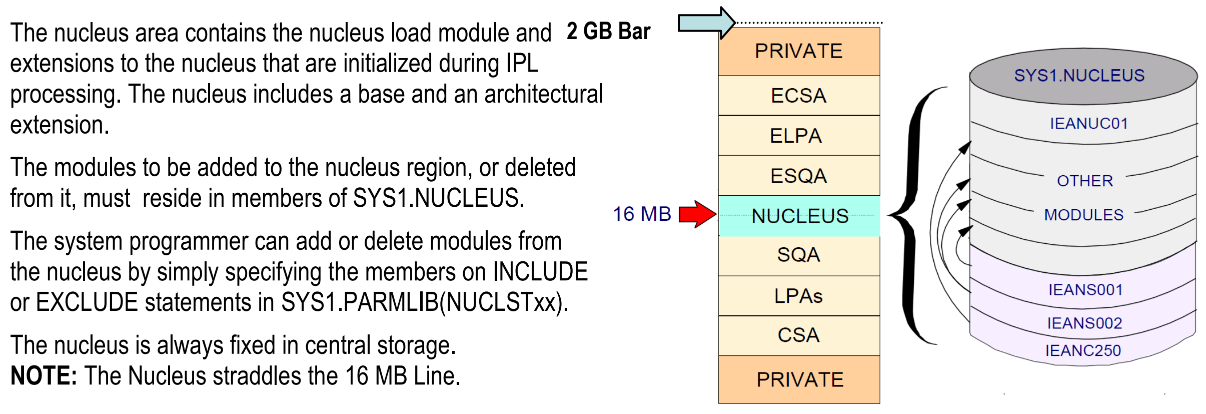
   Specified by: A list of modules to be put in FLPA must be established by the installation in the fixed LPA list (IEAFIXxx) member of SYS1.PARMLIB. Modules from any partitioned data set can be included in the FLPA. FLPA is selected through specification of the FIX system parameter in IEASYSxx or from the operator's console at system initialization. Any module in the FLPA will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected any library named in IEAFIXxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library. This area may be used to contain reenterable routines from either APF-authorized or non-APF-authorized libraries that are to be part of the pageable extension to the link pack area during the current IPL.

3. **Modified Link Pack Area (MLPA/Extended MLPA)** This area may be used to contain reenterable routines from either APF-authorized or non-APF-authorized libraries that are to be part of the pageable extension to the link pack area during the current IPL. Any module in the modified link pack area will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected any library named in IEALPAxx to avoid system security and integrity exposures, just as you would protect any APF-authorized library. The MLPA exists only for the duration of an IPL. Therefore, if an MLPA is desired, the modules in the MLPA must be specified for each IPL (including quick start and warm start IPLs). The MLPA is allocated just below the FLPA (or the PLPA, if there is no FLPA); the extended MLPA is allocated above the extended FLPA (or the extended PLPA if there is no extended FLPA). When the system searches for a routine, the MLPA is searched before the PLPA.

   Note: Loading a large number of modules in the MLPA can increase fetch time for modules that are not loaded in the LPA. This could affect system performance. The MLPA can be used at IPL time to temporarily modify or update the PLPA with new or replacement modules. No actual modification is made to the quick start PLPA stored in the system's paging data sets. The MLPA is read-only, unless NOPROT is specified on the MLPA system parameter.

System Queue Area (SQA-Fixed) SQA is allocated in fixed storage upon demand as long-term fixed storage and remains so until explicitly freed. The number of central frames assigned to SQA may increase and decrease to meet the demands of the system. All SQA requirements are allocated in 4K frames as needed. These frames are placed within the preferred area (above 16 megabytes, if possible) to keep long-term resident pages grouped together. If no space is available within the preferred area, and none can be obtained by stealing a non-fixed/unchanged page, then the "reconfigurable area"† is reduced by one storage increment and the increment is marked as preferred area storage. An increment is the basic unit of physical storage. If there is no "reconfigurable area" to be reduced, a page is assigned from the V=R area. Excluded from page stealing are frames that have been fixed (for example, through the PGFIX macro), allocated to a V=R region, placed offline using a CONFIG command, have been changed, have I/O in progress, or contain a storage error.

Nucleus Area The nucleus area contains the nucleus load module and extensions to the nucleus that are initialized during IPL processing. The nucleus includes a base and an architectural extension. Specify the correct architectural extension with the ARCHLVL statement in the LOADxx member of SYS1.PARMLIB for your system to run in either ESA/390 mode or z/Architecture mode.


The nucleus area contains the nucleus load module and extensions to the nucleus that are initialized during IPL processing. The nucleus includes a base and an architectural extension.

The modules to be added to the nucleus region, or deleted from it, must reside in members of SYS1.NUCLEUS.

The system programmer can add or delete modules from the nucleus by simply specifying the members on INCLUDE or EXCLUDE statements in SYS1.PARMLIB(NUCLSTxx).

The nucleus is always fixed in central storage.
NOTE: The Nucleus straddles the 16 MB Line.

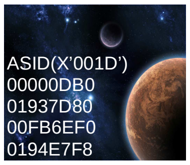
Above the 2 GB Bar: The real storage 2 GB limit was broken with z/Architecture. OS/390 V2R10 and z/OS V1R1 support up to 128 GB of real storage when running in z/Architecture mode on an IBM 2064. z/OS V1R2 provides for up to 256 GB of central storage to be configured to a z/OS image. As previously mentioned, z/Architecture broke the 2 GB (31-bit) central storage limit and the 2 GB (31-bit) address limit, and moved the limit to 16 Exa (64-bit).

The maximum of a z/OS address space is 16 Exa addresses, which makes the new address space 8 billion times the size of the former 2 G address space. However, any new created address space in z/OS is initialized with 2 G addresses (as it was previously), but with the potential to go beyond.

For compatibility, the layout of the virtual storage areas for an address space is the same below 2 GB. The area that separates the virtual storage area below the 2 G address from the user private area is called the *bar*, as shown in above figure and is 2 GB addresses thick. In a 64-bit virtual storage environment, the terms "above the bar" and "below the bar" are used to identify the areas between 2^{**31} and 2^{**64-1}, and 0 and 2^{**31-1}, respectively.

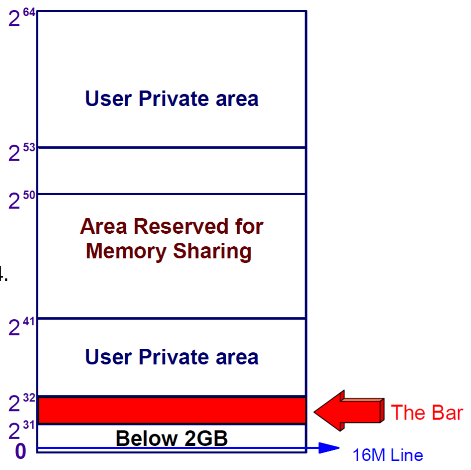
For example, a address in the range 0 to 7FFFFFFF is below the bar. An address in the range FFFFFFFF to 7FFFFFFF_FFFFFFFF is above the bar. This is basically an alteration to the 2 G 31-bit terminology that related "below the line" to 24-bit storage, and "above the line" to 31-bit addresses.

The 64-bit layout is next:



The 64-bit address space map is:

- **0 to 2**31:** The layout is the same (2 GB addressing).
- **2**31 to 2**32**: From 2 GB to 4 GB is considered the *bar*. Below the bar can be addressed with a 31-bit address. Above the bar requires a 64-bit address. Just as the system does not back the page at 7FFFF000 in order to protect programs from addresses which can wrap to 0, the system does not back the virtual area between 2 GB and 4 GB. That means a 31-bit address with the high bit on will always program check if used in AMODE 64.
- **2**31 - 2**41:** The Low Non-shared area starts at 4G and goes to 2**41.
- **2**41 - 2**50:** The Shared Area starts at 2**41 and goes to 2**50 or higher if requested (up to **2 **53**).
- **2**50 - 2**64:** The High Non-shared area starts at 2**50 or wherever the Shared Area ends and goes to 2**64.



The ***shared area*** may be shared between programs running in private areas from specific address spaces. In contrast, the below the bar common area is shared between all address spaces.

The area above the bar is designed to keep data (such as DB2 buffer pool and WebSphere data), and not to load modules (programs). There is *no* RMODE64 as a load module attribute. However, such programs running below the bar may request virtual storage above the bar and access it. In order to access such an address, the program must be AMODE64.

To allocate and release virtual storage above 2 GBs, a program must use the services provided in the **IARV64 macro**. The GETMAIN, FREEMAN, STORAGE, and CPOOL macros do not allocate storage above the 2 GB address, nor do callable cell pool services.

User Private area

The area above the bar is intended for application data; no programs run above the bar. No system information or system control blocks exist above the bar, either. Currently there is no common area above the bar.

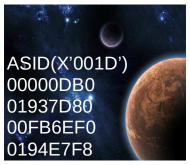
The *user private area*, as shown directly above includes:

- Low private: The private area below the line
- Extended private: The private area above the line
- Low Non-shared: The private area just above the bar
- High Non-shared: The private area above Shared Area

As users allocate private storage above the bar, it will first be allocated from the Low Non-shared area. Similarly, as the Shared Area is allocated, it will be allocated from the bottom up. This is done to allow applications to have both private and shared memory above the bar, and avoid additional machine cycles to perform dynamic address translation (DAT).

For virtual storage above the bar, a JCL keyword (**MEMLIMIT**) is introduced on the JOB and EXEC JCL statements. For virtual storage above the bar, there is no practical limit to the amount of virtual address range that an address space can request. However, there are practical limits to the central storage and auxiliary storage needed to back the request. Therefore, a limit is placed on the amount of usable virtual storage above the bar that an address space can use at any one time.

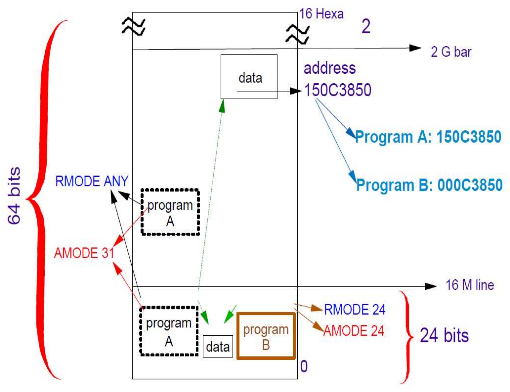
NOTE: MEMLIMIT controls the amount of usable storage above the 2 GB line. Also, there is an exit IEFUSI which does the same.



Size and number notation

Symbol	Power of 2	Decimal Value
Kilo (K)	2**10	1024
Mega (M)	2**20	1,048,576
Giga (G)	2**30	1,073,741,824
Tera (T)	2**40	1,099,511,627,776
Peta (P)	2**50	1,125,899,906,842,624
Exa (E)	2**60	1,152,921,504,606,846,976

Program Addressing Modes A program running on z/OS and Systemz mainframe can run with 24-, 31-, or 64-bit addressing (and can switch among these if needed). This is known as tri-modal addressing and it is unique in the industry. To address the high virtual storage available with the 64-bit architecture, the program uses 64-bit-specific instructions. Although the architecture introduces unique 64-bit exploitation instructions, the program can use both 31-bit and 64-bit instructions, as needed.



Addressing mode and residence mode
With the MVS/XA came the concept of *addressing mode* (AMODE), a program attribute to indicate which hardware addressing mode should be active to solve an address, that is, how many bits should be used for solving and dealing with addresses.
AMODE= 24: Indicates that the program may address up to 16 M virtual addresses.
AMODE= 31: Indicates that the program may address up to 2 G virtual addresses.
AMODE= 64: Indicates that the program may address up to 16-Exa virtual addresses (only in z/Architecture).
The concept of *residence mode* (RMODE), to indicate where a program should be placed in the virtual storage (by z/OS program management), when the system loads it from DASD:
RMODE=24: Indicates that the module must reside below the 16-MB virtual storage line. Among the reasons for RMODE24 we have: the program is AMODE24, the program has control blocks that must reside below the line.
RMODE= ANY: Indicates that the module might reside anywhere in virtual storage, but preferentially above the 16-MB virtual storage line. because of this such RMODE is also called RMODE 31. In z/OS there is not RMODE=64, because the virtual storage above 2G is not suitable for programs, only data AMODE and RMODE are load module attributes assigned when load modules are created by the Binder program and are placed in load module's directory entry in a partitioned data set.

NOTE: There is no load module attribute RMODE64.

– – –

† See #07 zNibbler (Dynamic Storage Reconfiguration)
Dynamic storage reconfiguration Dynamic storage reconfiguration allows central allocated to an LPAR to be changed while the LPAR is active. It is supported in LPARs running z/OS. Dynamic storage reconfiguration is not supported in coupling facility, z/VM, or Linux-Only LPARs. Dynamic storage reconfiguration provides the capability to reassign storage from one LPAR to another without the need to POR the CPC or IPL the recipient LPAR.

BONUS – see #61 (zOS' Subspace)