IBM z Systems

Does my address space have enough storage for me? An introduction to extended addressability

Session 17096

Peter Relson
IBM Poughkeepsie
relson@us.ibm.com
March 5, 2015

Ultra-short Presentation

Does my address space have enough storage for me?

Answer: Yes !!

Abstract

- This presentation will explore aspects of z/OS's virtual storage interfaces, concentrating on what is referred to as “extended addressability” (namely those functions not available in the “normal” 31-bit address space)

History

- Pre-MVS/XA – highest address was 2**24-1
- Pre-MVS/XA SP1.2 - cross-memory was introduced
- MVS/XA (SP2.1.0) – highest address is now 2**31-1
- MVS/ESA (SP3.1.0) – introduction of data spaces and AR mode
- z/Architecture (OS/390 R10) – highest real address is now (architecturally) 2**64-1
- z/Architecture (z/OS 1.3) – 64-bit virtual support

How much storage can my task access?

- **Pre-MVS/XA without cross-memory: 2**24 bytes**
- **Pre-MVS/XA with cross-memory: 2**24 bytes across your address space and however many spaces you can access at that time. Let's think of that as home + primary + secondary, so “3” spaces, and thus 3 * 2**24.**
- **MVS/XA – analogous, with 2**31 bytes for each of those address spaces, so 3 * 2**31.**
- **MVS/ESA – add in as many data spaces as can fit on your dispatchable unit and primary address space access lists (let's think of that as 512 * 2**31)**
- **z/Architecture – 64-bit virtual support. Data spaces are still limited to 2**31, but your address space could access 2**64 bytes (and you could access other address spaces' 2**64 bytes).**

But....

Of course there are limitations such as

- **the amount of real and auxiliary storage available (including Flash),**
- **Limits set by customer exit (typically overrideable by authorized programs) such as Region and Memlimit.**

Agenda

- 31-bit storage
- Data space (AR-mode) storage
- 64-bit storage
- Cross-memory (XM) access

How is the 31-bit Address Space storage organized?

Highest to Lowest

- **X'7FFFF000'-X'7FFFFFFF' not addressable by software convention**
- **Extended Private**
- **Extended CSA**
- **Extended LPA**
- **Extended SQA**
- **Extended read-write nucleus**
- **Extended read-only nucleus**
- **Read-only nucleus**
- **Read-write nucleus**
- **SQA**
- **LPA**
- **CSA**
- **Private**
- **Prefix Save Area (PSA)**

How do I manage 31-bit Address Space storage?

- Get/Obtain: GETMAIN (or STORAGE OBTAIN). Note that STORAGE OBTAIN,LINKAGE=SVC is the same as GETMAIN; STORAGE OBTAIN,LINKAGE=BRANCH is the same as GETMAIN BRANCH=YES
- Free/Release: FREEMAIN (or STORAGE RELEASE). STORAGE RELEASE,LINKAGE=SVC is the same as FREEMAIN; STORAGE RELEASE,LINKAGE=BRANCH is the same as FREEMAIN BRANCH=YES
- Pooled storage: CPOOL (you can also use CSRPGET et al)
- Attributes: storage key, fetch-protection, subpool, “ownership” (e.g., task)

How is a data space organized?

- Mostly organization of a data space is irrelevant – it is organized in whatever way the exploiter chooses (just as is the case with the organization of private storage)
- Data spaces are allocated in 4K blocks
- As of z/Architecture, Page 0 is always available
- But we recommend avoiding use of page 0 (due to the frequent error of using a pointer that is 0).
  - As of z/OS 2.1, when you create your data space, specify HIDEZERO=YES
  - Prior to z/OS 2.1 (this still works in z/OS 2.1), after you create your data space, use IARVSERV CHANGEACCESS,TARGET_VIEW=HIDDEN
- In both cases, accessing page 0 will blow up, after which you can debug your error

How do I manage data space storage?

- Create the data space: DSPSERV CREATE
- Delete the data space: DSPSERV DELETE
- Pooled storage: CSRPGET (et al)
- Attributes: storage key, fetch-protection

“Space” Terminology

- “Space Token” (STOKEN). An STOKEN is unique across all address spaces and all data spaces (past, current, and future). An Address Space's STOKEN can be found in field ASSBSTKN. A data space's STOKEN is returned by DSPSERV CREATE.
- PASN (Primary Address Space Number) == PASID (Primary Address Space ID = Primary ASID = P. CR4.48-63.
- HASN (Home ASN) = HASID = H. The initial dispatch space of a work unit. PSAAOLD → ASCBASID.
- SASN (Secondary ASN) = SASID = S. CR3.48-63

How do I access data space storage?

- **Access List and Access List Entry Token (ALET)**
  - **An ALET locates an entry in an Access List which identifies "which space" to access.**
  - **There are two types of access lists:**
    - **Primary Address Space Number Access List (PASN-AL). Every address space has one. Any work unit with that address space as its PASN can access it.**
    - **Dispatchable Unit Access List (DU-AL). Every work unit has one.**
  - **Add an entry to an access list and get an ALET back:**
    ALESERV {ADD | ADDPASN},STOKEN=s,ALET=a
    The ALET may reference an address space or a data space
  - **Remove an entry from an access list:**
    ALESERV DELETE,ALET=a

How do I use an access register?

- Set the ALET into the AR corresponding to a GR
- Be in “AR ASC Mode” (SAC X'200'). AKA “AR Mode”
- Use an instruction with that GR as a base register
- Special ALETs
  - 0 – primary address space
  - 1 – secondary address space. Could use MVCP, MVCS instead; suggest not using secondary ASC mode
  - 2 – home address space (software convention). Could use home ASC mode instead, if authorized
- AR does not apply when GR is used as an index register (references primary address space)
  - `L x,0(0,y)` y is base reg, AR y applies
  - `L x,0(y,0)` y is index reg, AR y does not apply

AR Mode Miscellany

- Don't forget the AR-mode linkage convention
  - Unless otherwise defined by the individual interface, users of Access Registers are responsible for preserving their caller's ARs 2-13.
- If you use any macros, be sure that SYSSTATE is set properly to indicate your AR mode state. For example, SYSSTATE ASCENV=AR vs SYSSTATE ASCENV=PRIMARY

How is the 64-bit Address Space storage organized?

Highest to Lowest

- **HV High private up to x'FFFFFFFF_FFFFFFFF'**
- **HV Shared**
- **HV Common**
- **HV Low private**
- **HV Local system area**
- **HV Storage for Java (2G-32G-1)**
- **31-bit address space (extended private, extended CSA, extended LPA, extended SQA, extended read-write nucleus, extended read-only nucleus, read-only nucleus, read-write nucleus, SQA, LPA,CSA, private, PSA)**

How do I manage high virtual (64-bit) storage?

- Allocate storage: IARV64 GETSTOR | GETCOMMON | GETSHARED – 1M units
  - IARST64 – smaller units, still power-of-2
- Delete the storage: IARV64 DETACH
- Pooled storage: CSRC4GET et al, IARCP64
- Attributes: storage key, fetch-protection, dumping attribute

IARV64 REQUEST=GETSTOR

- COND={NO|YES}
- SEGMENTS / UNITS (and UNITSIZE)
- PAGEFRAMESIZE
- TYPE={PAGEABLE | DREF | FIXED}
- ORIGIN=o
- KEY=k
- FPROT={YES|NO}
- MEMLIMIT={YES|NO|COND}
- DUMP={LIKERGN | LIKELSQA | NO}
- CONTROL={UNAUTH | AUTH}
- GUARDSIZE=g, GUARDSIZE64=g, GUARLOC={HIGH | LOW}
- TTOKEN=t
- Is it local system area?

IARV64 REQUEST=GETCOMMON

- COND={NO|YES}
- SEGMENTS / UNITS (and UNITSIZE)
- PAGEFRAMESIZE
- TYPE={PAGEABLE | DREF | FIXED}
- ORIGIN=o
- KEY=k
- FPROT={YES|NO}
- OWNERCOM={HOME | PRIMARY | SYSTEM | BYASID w/ OWNERASID}
- DUMP={LIKECSA | LIKESQA}
- GUARDSIZE=g, GUARDSIZE64=g, GUARDLOC={LOW | HIGH}

Other IARV64 functions

- DETACH – free a memory object
- PAGEFIX / PAGEUNFIX
- PCIEFIX / PCIEUNFIX – fix for PCIE I/O
- PAGEOUT / PAGEIN – page out to auxiliary storage
- DISCARDDATA – “release” the data
- CHANGEGUARD – change amount, attributes
- LIST – information about memory objects
- COUNTPAGES – 4K pages backing the virtual range
- GETSHARED – HV that can be shared by multiple users
- SHAREMEMOBJ – gain access to shared HV
- CHANGEACCESS – VIEW={READONLY | SHAREWRITE | HIDDEN}
- PROTECT / UNPROTECT – PROTECT makes it read-only

IARST64

Fast service to deal with storage in smaller chunks than the 1M minimum of IARV64 (the system rounds up to a power of two)

- **Register-only interface (no parameter list). It clobbers just about every register unless caller provides a savearea and uses REGS=SAVE**
- **Get or Free**
- **Common=NO**
  - **OwningTask, MEMLIMIT applies, Is it LocalSysArea?**
- **Common=YES**
  - **Owner**
- **Attributes**
  - **FPROT, Pageable | Dref | Fixed, Key, Abend or RC**

IARCP64

Simple pool services for cell sizes <= 520192 (the system rounds up to a power of two)

- **BUILD the pool (options similar to IARV64 REQUEST=GETSTOR to define storage attributes)**
  - **The cells may be defined with a “trailer” area that allows for setting on GET and checking on FREE**
- **GET an element (you may allow the pool to expand or not)**
- **FREE an element**
- **DELETE the pool**

AMODE 64 miscellany

- To get into AMODE 64: SAM64
- To get back to AMODE 31: SAM31
- To test your AMODE: TAM instruction
- If entered in AMODE 31 and then switching to AMODE 64, don't forget to clear the high halves of any base regs that might be used in that AMODE. For example, LLGTR 12,12
- Don't forget the 64-bit GR linkage convention
  - Unless otherwise defined by the individual interface, users of 64-bit GRs are responsible for preserving their caller's GR high halves 2-14
- If you use any macros, be sure that SYSSTATE is set properly to indicate your AMODE 64 state. For example, SYSSTATE AMODE64=YES vs SYSSTATE AMODE64=NO

AMODE 64 miscellany (cont)

- An AMODE 64 routine should not rely on register 15 on entry containing the entry point address (and it will not when control is passed via such interfaces as ATTACH and LINK); it should use relative branching.

AMODE 64 Considerations

- When using an interface, be sure to check that it supports not only your AMODE but also the type of storage you are intending to provide. If the interface does not mention AMODE 64 and 64-bit storage, it's wise not to assume that the service accepts 64-bit storage even if it supports AMODE 64. Use of “any” for AMODE may indicate the pre-z/Architecture meaning of “any” (namely, 24- or 31-bit).
  - I encourage you to bring to ID's attention any AMODE 64 service that does not make it clear whether data can be above 2G.
- My guess is that more services currently support data in data spaces (input 31-bit address plus ALET) than support data above 2G. It is often easy to upgrade a service to allow an AMODE 64 caller but considerably more difficult for that service to support data above 2G.

XM terminology

- PC instruction may be space-switch (changes PASN) or non-space-switch (PASN remains unchanged)
  - Secondary may be set to old-PASN or new-PASN
- Cross-Memory mode (XM) when it is not true that H=P=S
  - Work unit starts with H=P=S, lets say “27”
  - Issues space-switch PC to 35. Now H=27, P=35, S=27 (when S is set to old-PASN which was 27)
  - Issues space-switch PC to 42. Now H=27, P=42, S=35
- PR instruction (or PT when not-stacking)

XM terminology (cont)

- Key-mask: a 16-bit area in which each bit corresponds to the corresponding key (bit 0 to key 0, etc.)
- PSW-key mask (PKM). 16-bits in CR3. PKM is ignored if in supervisor state. It can confer additional authority.
  - For example: If you are “key 8” problem state but your PKM has the “key 0” bit on, then you are allowed to switch to PSW key 0.
  - z/OS must be very careful in managing the PKM to make sure that unwanted bits do not get set.
- There are also “AKM” (Authorization key mask) and “EKM” (Entry key mask)

XM terminology (cont)

- Linkage Table (also Linkage First Table, Linkage Second Table, Linkage Index or LX): involved in resolving a PC number to its definition (including the target routine)
- Entry Table (also Entry Table Entry, Entry Index or EX): involved in resolving a PC number to its definition
- PC Number: consists of 12-/16-/24-bit LX, 8-bit EX and (optionally) 32-bit sequence number

Basic PC vs Stacking PC interface

- Basic PC interface requires caller to save regs, and requires target to use PCLINK
- Stacking PC uses the linkage stack
- Prior to z/OS 1.6, many routines used non-stacking PC (they had been created before there was a linkage stack). In z/OS 1.6 most of the z/OS non-stacking PC's were changed to stacking.
- Macro expansions, to allow for compilation with “new” macros but execution on “old” releases remained unchanged unless told “OK to rely on z/OS 1.6”

SYSSTATE OSREL

- Introduced in z/OS 1.6
- For example, SYSSTATE OSREL=ZOSV1R6
- Recommended to use the newest release that you can be sure accurately reflects your possible execution environments
- Sample LXRES expansion fragments
  - With SYSSTATE OSREL=ZOSV1R6: L / L / L / PC
  - Without: STM / ESAR / ST / L / L / L / PC / L / SSAR / LM / L

Authority Table (AT) and Authority Index (AX)

- Located by ASTE (which is located by control register or access list entry or entry table entry)
- 2 bits per entry (P for Primary, S for SSAR), uses AX (CR4.32-47)
  - P bit: Are you allowed to make this space your primary?
  - S bit: Are you allowed (with SSAR or SSAIR) to make this space your secondary?
- Also, for access list permission may use EAX (CR8.32-47)
- PC routine may define the EAX
- Authority Index (AX)
  - Special value 0: neither Primary nor SSAR authority
  - Special value 1: both Primary and SSAR authority
  - Other values assigned via AXRES service

XM services

- AXRES, AXSET, AXFRE, ATSET
- LXRES – reserve an LX
- ETDEF – define entry table entries for PC routines
- ETCRE – create an entry table
- ETCON – connect user to an entry table
- ETDIS – disconnect user from an entry table
- ETDES – destroy (delete) an entry table
- LXFRE – free a previously-reserved LX
- PCLINK – applies only to non-stacking PC. Stay away if at all possible. Don't use non-stacking PC if at all possible.

XM services: AXRES

AXRES – reserve an Authorization Index (AX)

- **Typically not used if you are using a system LX**
- **AXRES AXLIST=al**
- **AXLIST=al identifies a list of halfwords that indicates the number of AX's to reserve (usually 1) and room to return the reserved AX's**

XM services: AXSET

AXSET – set the AX of home

- **AXRES AX=ax**
- **AX=ax specifies the new AX value**

XM services: AXFRE

AXFRE – free an Authorization Index (AX)

- **AXFRE AXLIST=al**
- **AXLIST=al identifies a list of halfwords that indicates the number of AX's to free and the AX's to free**

XM services: ATSET

ATSET – set an Authority Table entry for an Authority Index

- **ATSET AX=ax,PT={NO|YES},SSAR={NO|YES}**
- **AX=ax identifies the AX which in turn identifies the AT entry**
- **PT=p indicates how to set the Authority Table entry P bit**
- **SSAR=s indicates how to set the Authority Table entry S bit**

XM services: LXRES

LXRES – Reserve one or more Linkage Indexes (LX's)

- **LXRES {LXLIST=l | ELXLIST=el},**
  [LXSIZE={12|16|23|24},]
  [REUSABLE={NO|YES},]
  [SYSTEM={NO|YES},]
- **A system LX is automatically available to all address spaces (the entry table creator does the ETCON). All users of non-system LX must "connect".**
- **Z/OS 1.6 introduced the concept of a "reusable LX" and an LX bigger than 12 bits. Z/OS supports LX's up to 16 bits. But you should use LXSIZE=23 or 24 if you can, to accommodate future expansion.**
- **A "reusable LX" has an LX > 12 bits and also has a sequence number. When PC is issued, the PC number is used "normally" and the sequence number is placed in the high 32 bits of GR 15. This LX can be reused upon termination of the owner.**
- **If running on z/OS 2.1 or later, reusable LX is known to be available.**

XM services: LXRES (cont)

- For a non-reusable LX, reuse can still happen – if there are no more connectors. Until that happens, the LX is temporarily non-reusable. But if this was a system LX and there were space-switch entries, it is considered that there are always connectors and the LX is permanently non-reusable. However, for a system LX, a newly restarted “owner” can re-connect, allowing “continued” use.
- LXLIST=l identifies a list of fullwords that indicates the number of LX's to reserve and provides room to return the reserved LX's
- EXLIST=el identifies an area that starts with a fullword indicating the number to reserve and provides room to return the 8-byte extended LX's (consisting of a sequence number and the LX)

XM services: ETDEF

ETDEF – build/define an entry table descriptor (ETD)

- **The ETD consists of a header and one or more ETD entries**
- **ETDEF TYPE=INITIAL creates a static header**
- **ETDEF TYPE=ENTRY creates a static ETD entry**
- **ETDEF TYPE=FINAL terminates the static ETD**
- **If modification is needed, you would define a suitable area and copy the static ETD there before modifying.**
- **The address of the resulting ETD is input to ETCRE**
- **ETDEF TYPE=SET,ETEADR replaces fields in an existing (potentially copied) ETD entry (often used with the ROUTINE parameter)**
- **ETDEF TYPE=SET,HEADER changes the number of entries in an existing (potentially copied) ETD header**

XM services: ETDEF TYPE=ENTRY

ETDEF TYPE=ENTRY – define an ETD entry

- **[AKM=a] – identifies the authority key mask. E.g., AKM=(2,3,5:8)**
- **[ARR=arr] – provide the 8-char name in quotes, or the routine address. The system will attempt to locate the routine in LPA**
- **[ARRCOND={NO|YES}] – YES indicates that the ARR is conditional and should be ignored if found in an environment (FRRs) where ARRs are not allowed**
- **[ASYNCH={YES|NO}] – NO indicates that the ARR cannot be interrupted by asynchronous exits**
- **[CANCEL={YES|NO}] – NO indicates that the ARR cannot be interrupted by cancel**
- **[ASCMODE={PRIMARY|AR}] – the PC routine is to be given control in this ASC mode**

XM services: ETDEF TYPE=ENTRY (cont)

- [EAX=e] – identifies the EAX for the stacking PC
- [EK=ek] – the PC routine is to be given control in this key
- [EKM=ek] – identifies the entry key mask (EKM). E.g., EKM=(0,4:6). See PKM
- [PARM1=p1] – 4 bytes to be placed in first 4 bytes of “latent parameter”
- [PARM2=p2 – 4 bytes to be placed in second 4 bytes of latent parameter
- [PC=STACKING|BASIC] – the PC is stacking or basic
- [PROGRAM=pname | ROUTINE=raddr] – identifies the program name (the system will look for this in LPA or nucleus) or the routine address (see RAMODE)

XM services: ETDEF TYPE=ENTRY (cont)

- [PKM={OR|REPLACE}] – create the PKM by OR with the EKM or by replacing with the EKM
- [RAMODE={31|24|64}] – the AMODE when the routine address is provided
- [SASN={OLD|NEW}] – Upon the PC, set SASN to the OLD PASN or to the NEW PASN
- [SSWITCH={NO|YES}] – YES indicates that the PC is a space-switch PC, into the address space of the entry table creator
- [STATE={PROBLEM|SUPERVISOR}] – the PC routine is to get control in this ASC mode. If the PC can be issued in supervisor state, do not use STATE=PROBLEM. This could lead to unexpected undesirable results. z/OS does not support “authority decreasing” PC's (this is a not-well-known fact)

XM services: ETCRE

ETCRE – create an entry table

- **ETCRE ENTRIES=e**
- **ENTRIES=e identifies an area formed by ETDEF macro invocations that comprise the Entry Table Descriptor that is to be made into an entry table**
- **Register 0 on return from ETCRE is a token that is to be provided via the TKLIST parameter of ETCON**
- **The owner of an entry table must be non-swappable**

XM services: ETCON

ETCON – connect to an entry table

- **ETCON TKLIST=tk,**
  [LXLIST=l | ELXLIST=el]
- **TKLIST=tk identifies a list of fullwords**
  - **The first word is the number of entry tables to be connected**
  - **Each subsequent word is the output token from a preceding ETCRE**
- **LXLIST=l | ELXLIST=el identifies an area that is output from a corresponding LXRES service**

XM services: ETDIS

ETDIS – Disconnect from an entry table

- **ETDIS TKLIST=tk**
- **TKLIST=tk identifies a list of fullwords**
  - **The first word is the number of entry tables to be disconnected**
  - **Each subsequent word is the output token from a preceding ETCRE that indicates the entry table to disconnect**

XM services: ETDES

ETDES – Destroy an entry table

- **ETDES TOKEN=t[,PURGE={NO|YES}]**
- **TOKEN=t identifies the output token from a preceding ETCRE that indicates the entry table to disconnect**
- **PURGE=p specifies whether the entry table is first to be disconnected from all linkage tables before being destroyed. Unless PURGE=YES, if any outstanding connections exist, the caller of ETDES is abended.**

XM services: LXFRE

LXFRE – Free one or more LX's

- **LXFRE {LXLIST=l | ELXLIST=el}[,FORCE={NO|YES}]**
- **LXLIST=l | ELXLIST=el: identifies an area such as used on a previous LXRES request that identifies the LX's to be freed**
- **FORCE=YES indicates that the linkage index is to be freed even if entry tables are connected to it. Any connected entries are disconnected before the linkage index is freed.**

Create a server address space available to “some”

Setup

- **LXRES ELXLIST=elxl,REUSABLE=YES,SYSTEM=NO,LXSIZE=23**
- **AXRES AXLIST=axl**
- **AXSET AX=a**
- **ATSET AX=a,PT=YES,SSAR=YES**
- **ETDEF's to define theETD**
- **ETCRE ENTRIES=theETD**
- **ETCON TKLIST=TKL,ELXLIST=elxl**

Client would use:

- **ATSET, ETCON**

A server address space available to “some” (cont)

Cleanup

- **ETDIS TKLIST=TKL, ATSET AX=a,PT=NO,SSAR=NO**
- **ATSET AX=a,PT=NO,SSAR=NO**
- **ETDES TOKEN=t**
- **AXSET AX=ax_0**
- **AXFRE AXLIST=axl**
- **LXFRE ELXLIST=elxl**

Create a server address space available to “all”

Differences from “some”

- **Instead of LXRES with SYSTEM=NO, use SYSTEM=YES. If the intent is not to terminate (and if on abnormal termination you restart and re-connect), there is no need to use REUSABLE=YES**
- **Instead of AXRES, AXSET with returned AX, and ATSET, use AXSET to set to an AX of 1:**

```
LHI   0,1
AXSET AX=(0)
```

- **For cleanup, there is no need to ATSET and AXFRE, but you should AXSET to AX of 0:**

```
SLR   0,0
AXSET AX=(0)
```

Create a server address space available to “all”

Setup

- **AXSET AX=ax_1**
- **LXRES ELXLIST=elxl,REUSABLE=YES,SYSTEM=YES,LXSIZE=23**
- **ETDEF's to define theETD**
- **ETCRE ENTRIES=theETD**
- **ETCON TKLIST=tkl,ELXLIST=elxl**

Cleanup

- **AXSET AX=ax_0**

If the server terminates and restarts, it would issue AXSET for the new space and re-issue ETCON (and thus must make sure that the TKLIST and ELXLIST areas are accessible). It would not re-issue LXRES.

Example: Server “some”: Static Data

```
@STAT     DS    0D
ETD_Stat ETDEF TYPE=INITIAL
ETD0     ETDEF TYPE=ENTRY,AKM=(0:15),                                  *
               ASCMODE=PRIMARY,EK=8,                                   *
               ROUTINE=1, PROGRAM=MYMOD,ARR='MYMODARR',                *
               PKM=OR,EKM=8,                                           *
               PC=STACKING,SASN=OLD,                                   *
               SSWITCH=NO,STATE=SUPERVISOR
* ETD1 will be modified using ETDEF TYPE=SET, so
* provides the minimum to make the macro happy
ETD1     ETDEF TYPE=ENTRY,                                             *
               ROUTINE=0,AKM=0
         ETDEF TYPE=FINAL
ETD_Stat_Len EQU *-ETD_Stat
LXRESLS  LXRES MF=L
ETCONLS  ETCON MF=L
ETDESLS  ETDES MF=L
LXFRELS  LXFRE MF=L
AX0      DC    H'0'
         LTORG
*
```

Example: Server “some”: Dynamic Data

```
@DYN      DSECT
ETD_Dyn   DS    CL(ETD_Stat_Len)
ETD1_Dyn  EQU   ETD_Dyn+(ETD1-ETD_Stat)
TermECB   DS    F
AXL       DS    2H        AXList
AX        EQU   AXL+2     AX
TKL       DS    2F        TKList
TK        EQU   TKL+4     Token
ELXL      DS    3F        Extended LXList
LXSeq#    EQU   ELXL+4    LX Seq#
LX        EQU   ELXL+8    LX
LXRESL    LXRES MF=L
          ORG   LXRESL
ETCONL    ETCON MF=L
          ORG   LXRESL
ETDESL    ETDES MF=L
          ORG   LXRESL
LXFREL    LXFRE MF=L
          ORG   ,
@DYNSIZE  EQU   *-@DYN
```

Example: Server “some”: Module Start

```
SERVSOME CSECT
SERVSOME AMODE 31
SERVSOME RMODE 31
         IEABRCX DEFINE
         IEABRCX ENABLE
         SYSSTATE OSREL=SYSSTATE,ARCHLVL=OSREL
         BSM   14,0
         BAKR  14,0
         LARL  12,@STAT
         USING @STAT,12
         SLR   13,13
         MODESET MODE=SUP
         STORAGE OBTAIN,LENGTH=@DYNSIZE,CALLRKY=YES
         LR    11,1
         USING @DYN,11
         MVHI  TermECB,0
```

Example: Server “some”: LXRES, AXRES, AXSET

```
MVHI   ELXL,1
MVC    LXRESL,LXRESLS
LXRES  ELXLIST=ELXL,LXSIZE=23,SYSTEM=NO,                    *
       MF=(E,LXRESL)
LTR    15,15
JNZ    IssueAbend
MVC    AXL(2),=H'1'
AXRES  AXLIST=AXL
LTR    15,15
JNZ    IssueAbend
AXSET  AX=AX
LTR    15,15
JNZ    IssueAbend
LA     0,ETD_Dyn
LA     14,ETD_Stat
LA     15,ETD_Stat_Len
LR     1,15
MVCL   0,14
```

Example: Server “some”: ETDEF, ETCRE, ATSET, ETCON

```
LARL   2,ETD1_Routine
LARL   3,ETD1_Routine_ARR
ETDEF  TYPE=SET,ETEADR=ETD1_Dyn,                              *
       AKM=0,                                                 *
       ROUTINE=(2),RAMODE=31,ARR=(3),                         *
       ASCMODE=PRIMARY,EK=8,                                  *
       PKM=OR,EKM=8,                                          *
       PC=STACKING,SASN=OLD,                                  *
       SSWITCH=YES,STATE=SUPERVISOR
ETCRE  ENTRIES=ETD_Dyn
LTR    15,15
JNZ    IssueAbend
ST     0,TK
ATSET  AX=AX,PT=YES,SSAR=YES
LTR    15,15
JNZ    IssueAbend
MVHI   TKL,1
MVC    ETCONL,ETCONLS
ETCON  TKLIST=TKL,ELXLIST=ELXL,                               *
       MF=(E,ETCONL)
LTR    15,15
JNZ    IssueAbend
*
```

Example: Server “some”: Sample Issuing PC

```
* Example of forming PC# from LX and EX, using Seq#
         L     15,LX                Get LX
         OILL  15,X'01'             Apply EX
         LMH   15,15,LXSeq#
         PC    0(15)
* Server might now wait to learn of need to cleanup
         WAIT  ECB=TermECB
```

Example: Server “some”: Cleanup

```
ETDIS TKLIST=TKL
LTR   15,15
JNZ   IssueAbend
ATSET AX=AX,PT=NO,SSAR=NO
LTR   15,15
JNZ   IssueAbend
MVC   ETDESL,ETDESLS
ETDES TOKEN=TK,                                          *
      MF=(E,ETDESL)
LTR   15,15
JNZ   IssueAbend
AXSET AX=AX0
LTR   15,15
JNZ   IssueAbend
AXFRE AXLIST=AXL
LTR   15,15
JNZ   IssueAbend
MVC   LXFREL,LXFRELS
LXFRE ELXLIST=ELXL,                                      *
      MF=(E,LXFREL)
LTR   15,15
JNZ   IssueAbend
*
```

Example: Server “some”: End of module, etc

```
         STORAGE RELEASE,ADDR=(11),LENGTH=@DYNSIZE
         DROP  11
         MODESET MODE=PROB
         PR
IssueAbend ABEND 1
ETD1_Routine DS  0H
         PR
ETD1_Routine_ARR DS 0H
         BR    14
```

Summary

- There are lots of storage options available to you
- 31-bit services
- AR-mode services
- 64-bit services

And you may get to play with storage across multiple address spaces via XM services.

You will likely have plenty of storage available to your application (customer-permitting); the decision of which form to use lies with you.

References

Publications

- **z/OS V2R1 MVS Assembler Services Reference**
- **z/OS V2R1 MVS Authorized Assembler Services Reference**
- **z/OS V2R1 MVS Extended Addressability Guide**
- **z/Architecture Principles of Operation**

Questions?