GC28-1158-1
File No. S370-36

Program Product

MVS/Extended Architecture System Programming Library: 31-Bit Addressing

MVS/System Product - JES3 Version 2	5665-291
MVS/System Product - JES2 Version 2	5740-XC6

Second Edition (January, 1984)

This is a major revision of, and obsoletes, GC28-1158-0. See the Summary of Amendments following the Contents for a summary of the changes made to this manual.

This edition applies to the following program products and all subsequent releases until otherwise indicated in new editions or technical newsletters:

Assembler H Version 2 (5668-962)
Basic Telecommunications Access Method/System Product (BTAM/SP) (5665-279)
MVS/XA Data Facility Product (DFP) (5665-284)
Resource Measurement Facility (RMF) Version 3 (5665-274)
MVS/System Product - JES2 (MVS/SP) Version 2 (5740-XC6)
MVS/System Product - JES3 (MVS/SP) Version 2 (5665-291)
TSO Extensions (TSO/E) for MVS/XA (5665-285)

See the Summary of Amendments following the Contents for a summary of the changes made to this manual. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

Preface

This book is intended for programmers who are:

- Writing new assembler language programs to execute on MVS/Extended Architecture (MVS/XA)
- Changing existing assembler language programs, if necessary, to enable them to execute in an MVS/XA 31-bit addressing environment

Some of the guidelines and suggested coding practices will be useful to applications programmers. The book contains more technical detail than programmers writing in higher level languages require.

You should read this book before writing new programs or modifying existing programs to use 31-bit addresses. It defines terms used in 31-bit addressing and contains the following chapters:

- Chapter 1 - Understanding 31-Bit Addressing
- Chapter 2 - Planning for 31-Bit Addressing
- Chapter 3 - Addressing Mode and Residency Mode
- Chapter 4 - Establishing Linkage
- Chapter 5 - Performing I/O in 31-Bit Addressing Mode
- Chapter 6 - Understanding the Use of Real Storage

This book relies on the *MVS/Extended Architecture Conversion Notebook*, GC28-1143, for lists of changed services and control blocks.

This book also refers to:

MVS/Extended Architecture Supervisor Services and Macro Instructions, GC28-1114

MVS/Extended Architecture System Programming Library: System Macros and Facilities Volumes 1 and 2, GC28-1150 and GC28-1151

MVS/Extended Architecture System-Data Administration, GC26-4010

MVS/Extended Architecture Data Administration: Macro Instruction Reference, GC26-4014

MVS/Extended Architecture VSAM Administration Guide, GC26-4015

MVS/Extended Architecture System Programming Library: Service Aids, GC28-1159

Assembler H Version 2 Application Programming: Language Reference, GC26-4037

MVS/Extended Architecture Linkage Editor and Loader User's Guide, GC26-4011

370-Extended Architecture: Principles of Operation, GA22-7085

Contents

Figures

Summary of Amendments

Summary of Amendments
for GC28-1158-1
for MVS/System Product Version 2 Release 1.2

Changes to this publication include technical and editorial updates in support of MVS/Extended Architecture Release 1.2.

These changes include support for VSAM 31-bit addressing and new names for some Data Facility Product publications.

Summary of Amendments
for GC28-1158-0
as Updated June 30, 1983
by Technical Newsletter GN28-5099

A change to Data Management Access Methods for VSAM requirements has been made.

Chapter 1 - Understanding 31-Bit Addressing

MVS Extended Architecture (MVS/XA) supports 31-bit real and virtual addresses, which provide a maximum real and virtual address of two gigabytes (2^{31}) minus one. For compatibility with existing programs, MVS/XA also supports 24-bit real and virtual addresses. The basic changes in the system that provide for both 31-bit addresses and the continued use of 24-bit addresses are:

- A virtual storage map of two gigabytes with control program services to support programs executing or residing anywhere in virtual storage.
- Two new program attributes that specify expected address length on entry and intended location in virtual storage.
- **Bimodal operation**, a capability of the processor that permits the execution of programs with 24-bit addresses as well as programs with 31-bit addresses.
- New and changed instructions that are sensitive to addressing mode.

Virtual Storage

In the MVS/XA virtual storage map:

- Each address space has its own two gigabytes of virtual storage.
- Each major system area and the private area has a portion below the 16 megabyte line and an extended portion above the line but, logically, each of these areas can be thought of as one area.

Figure 1 shows the virtual storage map.

Addressing Mode and Residency Mode

In MVS/370, addresses are 24 bits long and programs can only reside in the area addressable by 24-bit addresses (below 16 megabytes). In MVS/XA, the processor can treat addresses as having either 24 or 31 bits. Addressing mode (AMODE) describes whether the processor is using 24-bit or 31-bit addresses. In MVS/XA programs can reside in 24-bit addressable areas, as with MVS/370, or beyond the 24-bit addressable area (above 16 megabytes). Residency mode (RMODE) specifies whether the program should reside in the 24-bit addressable area or if it can reside anywhere in 31-bit addressable storage.

Addressing mode (AMODE) and **residency mode** (RMODE) are program attributes specified (or defaulted) for each CSECT, load module, and load module alias. These attributes are the programmer's specification of the addressing mode in which the program is expected to get control and where the program is expected to reside in virtual storage.

AMODE defines the addressing mode (24, 31, or ANY) in which a program expects to receive control. Addressing mode refers to the address length that a program is prepared to handle on entry: 24-bit addresses, 31-bit addresses, or both (ANY). Programs with an addressing mode of ANY have been designed to receive control in either 24- or 31-bit addressing mode.

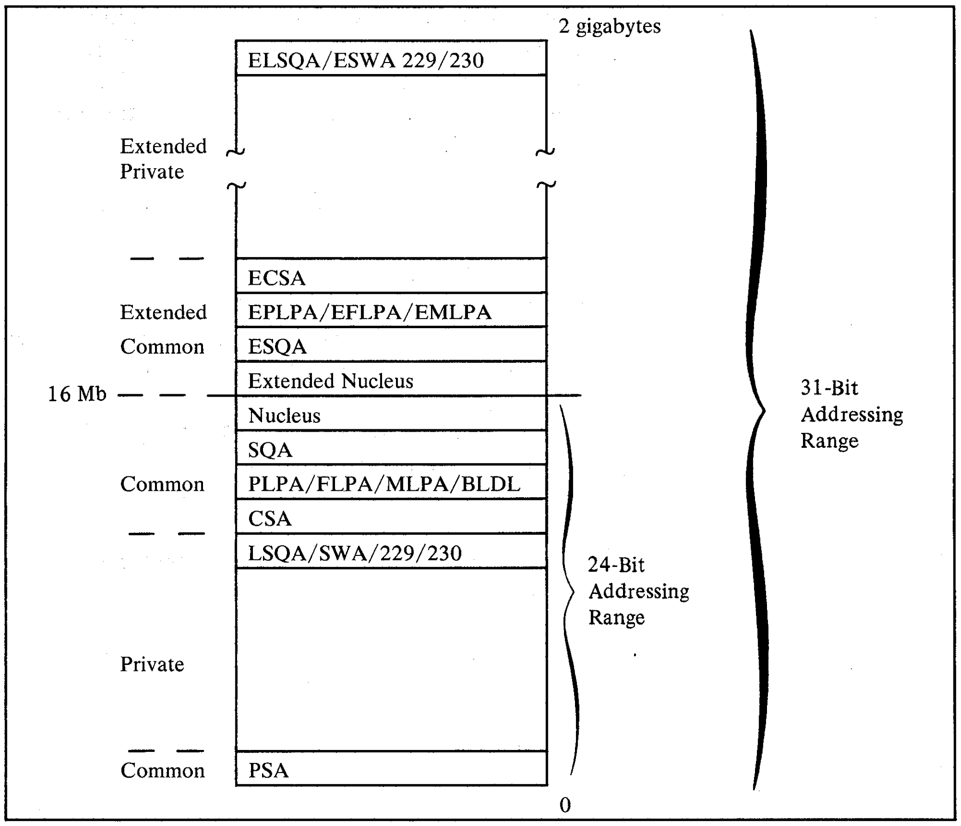


Figure 1. Two Gigabyte Virtual Storage Map

A 370-XA processor can operate with either 24-bit addresses (16 megabytes of addressability) or 31-bit addresses (2 gigabytes of addressability). A 370-XA processor uses PSW bit 32 (the A-mode bit) to control the length of addresses. When bit 32 is 0, the 370-XA processor operates in 24-bit addressing mode. When bit 32 is 1, the processor operates in 31-bit addressing mode. This ability of the processor to permit the execution of programs in 24-bit addressing mode as well as programs in 31-bit addressing mode is called **bimodal operation**. A program's AMODE attribute determines whether the program is to receive control with 24-bit or 31-bit addresses. Once a program gets control, the program can change the AMODE if necessary.

Figure 2 illustrates the 370-XA PSW.

0R000TIE	Prot. Key	1MWP	S0	CC	Prog. Mask	0000 0000	A	Instruction Address
0	8	12	16	18	20	24	32	63

Figure 2. 370-XA PSW

In 24-bit addressing mode, the processor treats all virtual addresses as 24-bit values. This makes it **impossible** for a program in 24-bit addressing mode to address virtual storage with an address greater than 16,777,215 (16 megabytes) because that is the largest number that a 24-bit binary field can contain.

In 31-bit addressing mode, the processor treats all virtual addresses as 31-bit values.

The processor supports bimodal operation so that both new programs and most old programs can execute correctly. (The *Conversion Notebook* describes the kinds of programs that might not execute correctly.) Bimodal operation is necessary because certain coding practices in existing programs depend on 24-bit addresses. For example:

- Some programs use a 4-byte field for a 24-bit address and place flags in the high-order byte.
- Some programs use the LA instruction to clear the high-order byte of a register. (In 24-bit addressing mode, LA clears the high-order byte; in 31-bit addressing mode, it clears only the high-order bit.)
- Some programs depend on BAL and BALR to return the ILC (instruction length code), the CC (condition code), and the program mask. (In 24-bit addressing mode they do; in 31-bit addressing mode they do not.)

In the PDS (partitioned data set) directory, each load module and each alias entry has an AMODE attribute.

A CSECT can have only one AMODE, which applies to all its entry points. Different CSECTs in a load module can have different AMODEs.

RMODE specifies where a program is expected to reside in virtual storage. The RMODE attribute is not related to real storage requirements. (RMODE 24 indicates that a program is coded to reside in virtual storage below 16 megabytes. RMODE ANY indicates that a program is coded to reside anywhere in virtual storage.)

In the PDS directory, each load module and each alias entry has an RMODE attribute. The alias entry is assigned the same RMODE as the main entry.

The following kinds of programs must reside in the range of addresses below the 16 megabyte line (addressable by 24-bit callers):

- Programs that have the AMODE 24 attribute
- Programs that have the AMODE ANY attribute
- Programs that use system services that require their callers to be AMODE 24
- Programs that use system services that require their callers to be RMODE 24
- Programs that must be addressable by 24-bit addressing mode callers

Programs without these characteristics can reside anywhere in virtual storage.

Chapter 3 describes AMODE and RMODE processing and MVS/XA support of AMODE and RMODE in detail.

Requirements for Execution in 31-Bit Addressing Mode

In general, to execute in 31-bit addressing mode a program must:

- Be assembled using Assembler H Version 2 and the MVS/XA macro instruction library.
- Be link edited using the linkage editor supplied with Data Facility Product (DFP) or be loaded using the loader supplied with DFP.
- Execute on an MVS/XA system.

Rules and Conventions for 31-Bit Addressing

It is important to distinguish the rules from the conventions when describing 31-bit addressing. There are only two rules, and they are associated with hardware:

1. The length of address fields is controlled by the A-mode bit (bit 32) in the PSW. When bit 32=1, addresses are treated as 31-bit values. When bit 32=0, addresses are treated as 24-bit values.

   Any data passed from a 31-bit addressing mode program to a 24-bit addressing mode program must reside in virtual storage below 16 megabytes. (A 24-bit addressing mode program cannot reference data above 16 megabytes without changing addressing mode.)

2. The A-mode bit affects the way some instructions work.

The conventions, on the other hand, are more extensive. Programs using MVS/XA services are expected to follow these conventions.

- A program should return control in the same addressing mode in which it received control.
- A program expects 24-bit addresses from 24-bit addressing mode programs and 31-bit addresses from 31-bit addressing mode programs.
- A program should validate the high-order byte of any address passed by a 24-bit addressing mode program before using it as an address in 31-bit addressing mode.

Mode Sensitive Instructions

The processor is sensitive to the addressing mode that is in effect (the setting of the PSW A-mode bit). The current PSW controls instruction sequencing. The instruction address field in the current PSW contains either a 24-bit address or a 31-bit address depending on the current setting of the PSW A-mode bit. For those instructions that develop or use addresses, the addressing mode in effect in the current PSW determines whether the addresses are 24 or 31 bits long.

Principles of Operation contains a complete description of the 370-XA instructions. The following topics provide an overview of the mode sensitive instructions.

BAL and BALR

BAL and BALR are addressing mode sensitive. In 24-bit addressing mode, BAL and BALR work the same way as they do when executed on a processor running in 370 mode. BAL and BALR put link information into the high-order byte of the first operand register and put the return address into the remaining three bytes before branching.

First operand register (24-bit addressing mode)

ILC	CC	PGM Mask	next sequential instruction address	
0	2	4	8	31

ILC - instruction length code
CC - condition code
PGM Mask - program mask

In 31-bit addressing mode, BAL and BALR put the return address into bits 1 through 31 of the first operand register and save the current addressing mode in the high-order bit. Because the addressing mode is 31-bit, the high-order bit is always a 1.

First operand register (31-bit addressing mode)

1	next sequential instruction address	
0	1	31

When executing in 31-bit addressing mode, BAL and BALR do not save the instruction length code, the condition code, or the program mask. IPM (insert program mask), a new 370-XA instruction, can be used to save the program mask and the condition code.

LA

The LA (load address) instruction, when executed in 31-bit addressing mode, loads a 31-bit value and clears the high-order bit. When executed in 24-bit addressing mode, it loads a 24-bit value and clears the high-order byte (as in 370 mode).

LRA

The LRA (load real address) instruction always results in a 31-bit real address regardless of the issuing program's AMODE. The virtual address specified is treated as a 24-bit or 31-bit address based on the value of the PSW A-mode bit at the time the LRA instruction is executed. The *Conversion Notebook* describes an additional difference.

Branching Instructions

BASSM (branch and save and set mode) and BSM (branch and set mode) are branching instructions that manipulate the PSW A-mode bit (bit 32). Programs can use BASSM when branching to modules that might have different addressing modes. Programs invoked via a BASSM instruction can use a BSM instruction to return in the caller's addressing mode. BASSM and BSM are described in more detail in Chapter 4.

BAS (branch and save) and BASR execute on extended architecture processors in either 370 or 370-XA mode. They:

- Save the return address and the current addressing mode in the first operand.
- Replace the PSW instruction address with the branch address.

The high-order bit of the return address indicates the addressing mode. BAS and BASR perform the same function that BAL and BALR perform in 31-bit addressing mode. In 24-bit mode, BAS and BASR put zeroes into the high-order byte of the return address register.

MVS/XA's Use of 31-Bit Addressing

MVS/XA, in addition to providing support for the use of 31-bit addresses by user programs, includes many system services that have been converted to use 31-bit addresses. Likewise, most new functions provided by MVS/XA use 31-bit addresses.

Some MVS/XA services are independent of the addressing mode of their callers. These services accept callers in either 24-bit or 31-bit addressing mode and use 31-bit parameter address fields. They assume 24-bit addresses from 24-bit addressing mode callers and 31-bit addresses from 31-bit addressing mode callers. Most supervisor macros are in this category.

Other MVS/XA services have restrictions with respect to address parameter values. Some of these services accept SVC callers and allow them to be in either 24-bit or 31-bit addressing mode. However, the same services might require branch entry callers to be in 24-bit addressing mode or might require one or more parameter addresses to be less than 16 megabytes.

Some services do not support 31-bit addressing at all. Among these are SPIE, STAE, SEGLD, SEGWT, and data management macros for all DFP access methods except VSAM [1]. (VSAM accepts entry by a program that executes in either 24-bit or 31-bit addressing mode.) The *Conversion Notebook* gives examples of the system services in each of these categories.

370-XA provides new instructions that support 31-bit addressing mode and bimodal operation. These new instructions are supported only by Assembler H Version 2 (5668-962) installed with the ADV or UNIV instruction set specified. The linkage editor functions that support MVS/XA are provided in Data Facility Product (DFP) (5665-284). Programs that are to execute in 31-bit addressing mode should be link edited using the DFP linkage editor or loaded using the DFP loader.

Location of Control Blocks

Some system control blocks have been moved above the 16 megabytes line. (The *Conversion Notebook* contains a list of the control blocks that were moved.) The topic "How to Change Addressing Mode" later in this book describes how 24-bit addressing mode programs can access data in control blocks that have moved above 16 megabytes.

[1] These access methods are SAM, BDAM, QISAM, ISAM, and BPAM.

Chapter 2 - Planning for 31-Bit Addressing

Most programs that run on MVS/370 will run on MVS/XA in 24-bit addressing mode without change. Some programs need to be modified to execute in 31-bit addressing mode to provide the same function on MVS/XA as on MVS/370. Still other programs need to be modified to run in 24-bit addressing mode. The *Conversion Notebook* helps you identify programs that need to be changed. This chapter helps you determine what changes to make to a module you are converting to 31-bit addressing and indicates what 31-bit address-related things to consider when writing new code.

Some reasons for converting to 31-bit addressing mode are:

- The program can use more virtual storage for tables, arrays, or additional logic.
- The program needs to reference control blocks that have been moved above the 16 megabyte line.
- The program is invoked by other 31-bit addressing mode programs.
- The program must run in 31-bit addressing mode because it is a user exit routine that the system invokes in 31-bit mode.
- The program needs to invoke system routines that expect to get control in 31-bit addressing mode.

Converting Existing Programs

Keeping in mind that 31-bit addressing mode programs can reside either below or above the 16 megabyte line, there are two ways to convert existing programs:

1. **Converting the program to use 31-bit addresses** - a change in addressing mode only.

   - You can change the entire module to use 31-bit addressing.
   - You can change only that portion that requires 31-bit addressing mode execution.

   Be sure to consider whether or not the code has any dependencies on 24-bit addresses. Such code does not produce the same results in 31-bit mode as it did in 24-bit mode. The topic "Mode Sensitive Instructions" in Chapter 1 contains an overview of instructions that function differently depending on addressing mode. The *Conversion Notebook* describes additional coding differences.

   Figure 3 summarizes the the things that you need to do to maintain the proper interface with a program that you plan to change to 31-bit addressing mode.

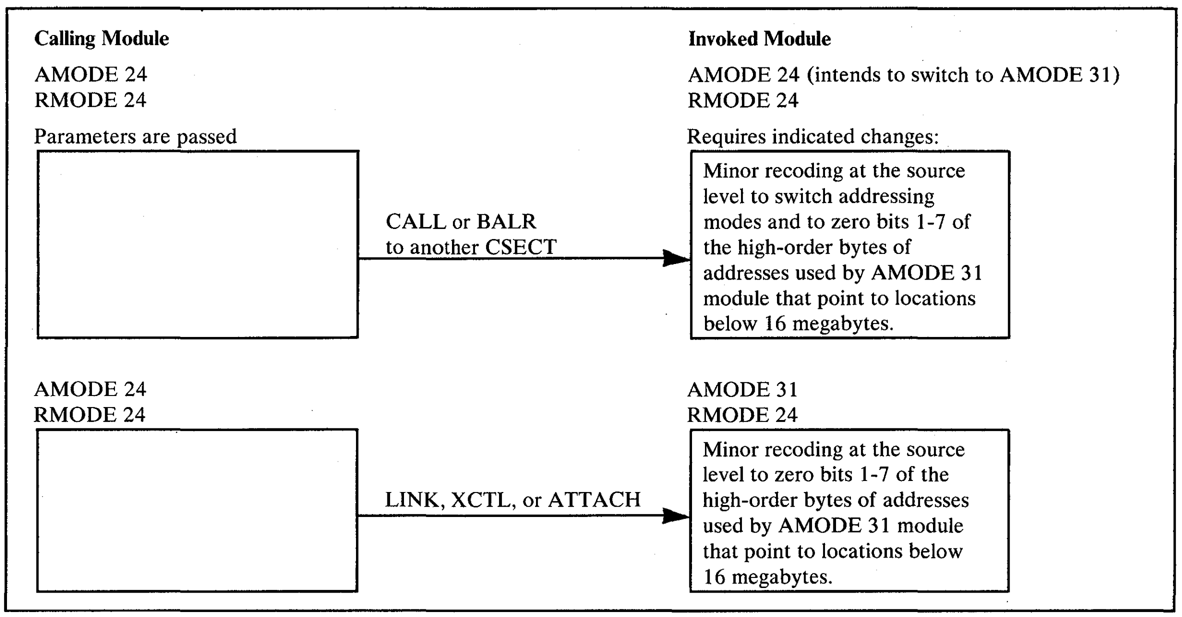

Calling Module
AMODE 24
RMODE 24
Parameters are passed
Invoked Module
AMODE 24 (intends to switch to AMODE 31)
RMODE 24
Requires indicated changes:
CALL or BALR to another CSECT
Minor recoding at the source level to switch addressing modes and to zero bits 1-7 of the high-order bytes of addresses used by AMODE 31 module that point to locations below 16 megabytes.
AMODE 24
RMODE 24
AMODE 31
RMODE 24
LINK, XCTL, or ATTACH
Minor recoding at the source level to zero bits 1-7 of the high-order bytes of addresses used by AMODE 31 module that point to locations below 16 megabytes.

Figure 3. Maintaining Correct Interfaces to Modules that Change to AMODE 31

2. **Moving the program above the 16 megabyte line** - a change in both addressing mode and residency mode

In general, the reason for moving an existing program above the 16 megabyte line is because there is not enough room for it below the line. For example:

- An existing program or application is growing so large that soon it will not fit below the 16 megabyte line.
- An existing application that now runs as a series of separate programs, or that executes in an overlay structure, would be easier to manage as one large program.
- Code is in the system area, and moving it would provide more room for the private area below 16 megabytes.

The techniques used to establish proper interfaces to modules that move above the 16 megabyte line depend on the number of callers and the ways they invoke the module. Figure 4 summarizes the techniques for passing control. The programs involved must ensure that any addresses passed as parameters are treated correctly. (High-order bytes of addresses to be used by a 31-bit addressing mode program must be validated or zeroed.)

Means of Entry to Moved Module (AMODE 31, RMODE ANY)	**Few AMODE 24, RMODE 24 Callers**	**Many AMODE 24, RMODE 24 Callers**
LOAD macro and BALR	• Have caller use LINK or • Have caller use LOAD macro and BASSM (invoked program returns via BSM) or • Change caller to AMODE 31, RMODE 24 before BALR	Create a linkage assist routine (described in Chapter 4). Give the linkage assist routine the name of the moved module.
BALR using an address in a common control block	• Have caller switch to AMODE 31 when invoking or • Change the address in the control block to a pointer-defined value (described in Chapter 4) and use BASSM. (The moved module will use BSM to return.)	Create a linkage assist routine (described in Chapter 4).
ATTACH, LINK, or XCTL	No changes required.	No changes required.
SYNCH in AMODE 24	• Have caller use SYNCH with AMODE=31 parameter or • Have caller switch to AMODE 31 before issuing SYNCH. • Change address in the control block to a pointer-defined value and use SYNCH with AMODE=DEFINED	Create a linkage assist routine (described in Chapter 4).

Figure 4. Establishing Correct Interfaces to Modules That Move Above 16 Megabytes

In deciding whether or not to modify a program to execute in 31-bit addressing mode either below or above the 16 megabyte line, there are several general considerations:

1. How and by whom is the module entered?

2. What system and user services does the module use that do not support 31-bit callers or parameters?

3. What kinds of coding practices does the module use that do not produce the same results in 31-bit mode as in 24-bit mode?

4. How are parameters passed? Can they reside above 16 megabytes?

Among the specific practices to check for are:

1. Does the module depend on the instruction length code, condition code, or program mask placed in the high order byte of the return address register by a 24-bit mode BAL or BALR instruction? One way to determine some of the dependencies is by checking all uses of the SPM (set program mask) instruction. SPM might indicate places where BAL or BALR were used to save the old program mask, which SPM might then have reset. The IPM (insert program mask) instruction can be used to save the condition code and the program mask.

2. Does the module use an LA instruction to clear the high-order byte of a register? This practice will not clear the high-order byte in 31-bit addressing mode.

3. Are any address fields that are less than 4 bytes still appropriate?

   Make sure that a load instruction does not pick up a 4-byte field containing a 3-byte address with extraneous data in the high-order byte. Make sure that bits 1-7 are zero.

4. Does the program use the ICM (insert characters under mask) instruction? The use of this instruction is sometimes a problem because it can put data into the high-order byte of a register containing an address, or it can put a 3-byte address into a register without first zeroing the register. If the register is then used as a base, index, or branch address register in 31-bit addressing mode, it might not indicate the proper address.

5. Does the program invoke 24-bit addressing mode programs? If so, shared data must be below 16 megabytes.

6. Is the program invoked by 24-bit or 31-bit addressing mode programs? Is the data in an area addressable by the programs that need to use it? (The data must be below 16 megabytes if used by a 24-bit addressing mode program.)

Writing New Programs for MVS/XA

You can write programs that execute in either 24-bit or 31-bit addressing mode on MVS/XA. However, in order to maintain an interface with existing programs and with some system services, your 31-bit addressing mode programs will need subroutines or portions of code that execute in 24-bit addressing mode. If your program resides below the 16 megabyte line, it can change to 24-bit addressing mode when necessary. If your program resides above 16 megabytes, it needs a separate load module to perform the linkage to an unchanged 24-bit addressing mode program or service. Such load modules are called linkage assist routines and are described in detail in Chapter 4.

When writing new programs for MVS/XA, there are some things you can do to simplify the passing of parameters between programs that might be in different addressing modes. In addition, there are new functions that you should consider and that you might need to accomplish your program's objectives. (In general, these new functions are not supported on MVS/370). Following is a list of suggestions for coding programs to run on MVS/XA:

- Use fullword fields for addresses even if the addresses are only 24 bits in length.

- When obtaining addresses from 3-byte fields in existing areas, use SR (subtract register) to zero the register followed by ICM (insert characters under mask) in place of the load instruction to clear the high-order byte. For example:

```
Rather than:    L     1,A

        use:    SR    1,1
                ICM   1,7,A+1
```

  The 7 specifies a 4-bit mask of 0111. The ICM instruction shown inserts bytes beginning at location A+1 into register 1 under control of the mask. The bytes to be filled correspond to the 1 bits in the mask. Because the high-order byte in register 1 corresponds to the 0 bit in the mask, it is not filled.

- If the program needs storage above 16 megabytes, obtain the storage using the VRU, VRC, RU, and RC forms of GETMAIN and FREEMAIN. These are the only forms that allow you to obtain and free storage above 16 megabytes. Do not use storage areas above 16 megabytes for save areas and parameters passed to other programs.

- Do not use the STAE macro; use ESTAE. STAE has 24-bit addressing mode dependencies.

- Do not use SPIE; use ESPIE. SPIE has 24-bit addressing mode dependencies.

- Do not use previous paging services macros; use PGSER.

- To make debugging easier, switch addressing modes only when necessary.

- Identify the intended AMODE and RMODE for the program in a prologue.

- Do not use the STAI parameter on the ATTACH macro; use the ESTAI parameter. STAI has 24-bit addressing mode dependencies.

- 31-bit addressing mode programs should use ESTAE or the ESTAI parameter on the ATTACH macro rather than STAE or STAI. When recovery routines refer to the PSW field in the SDWA, they should refer to SDWAEC1, which is the EC mode PSW at the time of error.

  User-written STAE and STAI routines need to be aware of the restricted support of the BC mode PSW fields in the SDWA. The instruction length and address fields contain zeroes in the following situations:

  - SDWACTL1 (BC mode PSW at time of error) contains zeroes in the designated fields when the error occurred while the program (or a service routine executing on behalf of the program) was executing in 31-bit addressing mode.

  - SDWACTL2 (BC mode PSW from the last program request block (PRB) on the request block (RB) chain) contains zeros when the last PRB on the RB chain refers to a program that was executing in 31-bit addressing mode.

When writing new programs for MVS/XA, you need to decide whether to use 24-bit addressing mode or 31-bit addressing mode.

The following are examples of kinds of programs that you should write in 24-bit addressing mode:

- Assembler language programs when you gain no extra value from making them execute in 31-bit addressing mode.

- Programs that must execute on MVS/370 as well as MVS/XA and do not require any new MVS/XA functions.

- Service routines, even those in the common area, that use system services requiring entry in 24-bit addressing mode or that must accept control directly from unchanged 24-bit addressing mode programs.

When you use 31-bit addressing mode, you must decide whether the new program should reside above or below 16 megabytes (unless it is so large that it will not fit below). Your decision depends on what programs and system services the new program invokes and what programs invoke it.

New Programs Below 16 Megabytes

The main reason for writing new 31-bit addressing mode programs that reside below the 16 megabyte line is to be able to address areas above 16 megabytes or to invoke 31-bit addressing mode programs while at the same time simplifying communication with existing 24-bit addressing mode programs or system services, particularly data management. For example, VSAM macro instructions accept callers in 24-bit or 31-bit addressing mode.

Even though your program resides below the 16 megabyte line, you must be concerned about dealing with programs that require entry in 24-bit addressing mode or that require parameters to be below 16 megabytes. Figure 9 in Chapter 4 contains more information about parameter requirements.

New Programs Above 16 Megabytes

When you write new programs that reside above the 16 megabyte line, your main concerns are:

- Dealing with programs that require entry in 24-bit addressing mode or that require parameters to be below 16 megabytes. Note that these are concerns of any 31-bit addressing mode program no matter where it resides.

- How routines that remain below the 16 megabyte line invoke the new program.

Writing Programs to Run on Both MVS/370 and MVS/XA

You can write new programs that will run on both MVS/370 and MVS/XA. If these programs do not need to use any new MVS/XA functions, the best way to avoid errors is to assemble the programs on MVS/370 with MVS/370 macro libraries. You can also assemble these programs on MVS/XA with the MVS/XA macro libraries, but you must generate MVS/370-compatible macro expansions by specifying the SPLEVEL macro instruction at the beginning of the programs.

If the program needs to use MVS/XA functions, your programming task is more difficult because most new MVS/XA functions are not supported on MVS/370. You need to use dual paths in your program so that on each system your program uses the services or macros that are supported on that system.

With minor exceptions, MVS/System Product Version 1 Release 3 programs that use published external interfaces as documented in the following publications will execute on MVS/System Product Version 2.

- *OS/VS2 MVS JCL*, GC28-0692

- *OS/VS2 Supervisor Services and Macro Instructions*, GC28-1114

- *OS/VS2 TSO Command Language Reference*, GC28-0646

- *OS/VS2 Guide to Writing a Command Processor or Terminal Monitor Program*, GC28-0648

- *OS/VS2 Data Management Macro Instructions*, GC26-3873
- *OS/VS2 Access Method Services*, GC26-3841
- *OS/VS Virtual Storage Access Method (VSAM) Programmers Guide*, GC26-3838

In addition to using published external interfaces, programs designed to execute on either system must use fullword addresses where possible and use no new functions on macro instructions except the LOC parameter on GETMAIN. These programs must also be aware of downward incompatible macros and use SPLEVEL as needed.

SPLEVEL Macro Instruction

Some macro instructions are **downward incompatible**. (The *Conversion Notebook* contains a list of these macros.) The level of the macro expansion (MVS/370 or MVS/XA) that is generated during assembly depends on the value of an assembler language global SET symbol. When the SET symbol value is 1, the system generates MVS/370 expansions. When the SET symbol value is 2, the system generates MVS/XA expansions.

The SPLEVEL macro instruction allows programmers to change the value of the SET symbol. The SPLEVEL macro instruction shipped with MVS/SP Version 2 sets a default value of 2 for the SET symbol. Therefore, unless a program or installation specifically changes the default value, the macros generated are MVS/XA macro expansions.

You can, within a program, issue the SPLEVEL SET=1 macro to obtain MVS/370(MVS/System Product Version 1 Release 3 Modification Level 0) expansions, or SPLEVEL SET=2 to obtain MVS/XA expansions. The SPLEVEL macro instruction sets the SET symbol value for that program's assembly only and affects only the expansions within the program being assembled. A single program can include multiple SPLEVEL macro instructions to generate different macro expansions. The following example shows how to obtain different macro expansions within the same program by assembling both expansions and making a test at execution time to determine which expansion to execute.

```
*   DETERMINE WHICH SYSTEM IS EXECUTING
         TM      CVTDCB,CVTMVSE (CVTMVSE is bit 0 in the
         BO      SP2            CVTDCB field. If bit 0=1,
                                it indicates that MVS/XA
                                is executing.)
*   INVOKE THE MVS/370 VERSION OF THE WTOR MACRO
         SPLEVEL SET=1
         WTOR    .....
         B       CONTINUE
SP2      EQU     *
*   INVOKE THE MVS/XA VERSION OF THE WTOR MACRO
         SPLEVEL SET=2
         WTOR    ......
CONTINUE EQU     *
```

Note: CVTMVSE is not defined prior to MVS/System Product Version 1 Release 3.

SPL: System Macros and Facilities and *Supervisor Services and Macro Instructions* describe the SPLEVEL macro instruction. The *Conversion Notebook* contains additional examples of its use.

Certain macro instructions produce a “map” of control blocks or parameter lists. These mapping macro instructions do not support the SPLEVEL macro. Mapping macros for different levels of MVS systems are available only in the macro libraries for each system. When programs use mapping macros, a different version of the program may be needed for each system.

Dual Programs

Sometimes two programs may be required; one for each system. In this case, use one of the following approaches:

- Keep each in a separate library
- Keep both in the same library but under different names

Chapter 3 - Addressing Mode and Residency Mode

Every program that executes in MVS/XA is assigned two program attributes: an addressing mode (AMODE) and a residency mode (RMODE). Programmers can specify these attributes for new programs. Programmers can also specify these attributes for old programs through reassembly, linkage editor PARM values, linkage editor MODE control statements, or loader PARM values. MVS/XA assigns default attributes to any program that does not have AMODE and RMODE specified.

Addressing Mode - AMODE

AMODE is a program attribute that can be specified (or defaulted) for each CSECT, load module, and load module alias. AMODE states the addressing mode that is expected to be in effect when the program is entered. AMODE can have one of the following values:

- **AMODE 24** - The program is designed to receive control in 24-bit addressing mode.
- **AMODE 31** - The program is designed to receive control in 31-bit addressing mode.
- **AMODE ANY** - The program is designed to receive control in either 24-bit or 31-bit addressing mode.

Residency Mode - RMODE

RMODE is a program attribute that can be specified (or defaulted) for each CSECT, load module, and load module alias. RMODE states the virtual storage location (either above the 16 megabyte line or anywhere in virtual storage) where the program should reside. RMODE can have the following values:

- **RMODE 24** - The program is designed to reside below the 16 megabyte line in virtual storage. MVS/XA places the program below 16 megabytes.
- **RMODE ANY** - The program is designed to reside at any virtual storage location, either above or below the 16 megabyte line. MVS/XA places the program above the 16 megabyte line unless there is no suitable virtual storage above 16 megabytes.

AMODE and RMODE Combinations

Figure 5 shows all possible AMODE and RMODE combinations and indicates which are valid.

AMODE and RMODE Combinations at Execution Time

At execution time, there are only three valid AMODE/RMODE combinations:

1. AMODE 24, RMODE 24, which is the default
2. AMODE 31, RMODE 24
3. AMODE 31, RMODE ANY

The ATTACH, LINK, and XCTL macro instructions give the invoked module control in the AMODE previously specified. However, specifying a particular AMODE does not guarantee that a module that gets control by other means will receive control in that AMODE. For example, an AMODE 24 module can issue a BALR to an AMODE 31, RMODE 24 module. The AMODE 31 module will get control in 24-bit addressing mode.

	RMODE 24	RMODE ANY
AMODE 24	Valid	Invalid (1)
AMODE 31	Valid	Valid
AMODE ANY	Valid (2)	It Depends (3)

(1.) This combination is invalid because an AMODE 24 module cannot reside above 16 megabytes.

(2.) This is a valid combination in that the assembler, linkage editor, and loader accept it from all sources. However, the combination is not used at execution time. Specifying ANY is a way of deferring a decision about the actual AMODE until the last possible moment before execution. At execution time, however, the module must execute in either 24-bit or 31-bit addressing mode.

(3.) The attributes AMODE ANY/RMODE ANY take on a special meaning when used together. (This meaning might seem to disagree with the meaning of either taken alone.) A module with the AMODE ANY/RMODE ANY attributes will execute on either an MVS/370 or an MVS/XA system if the module is designed to:

- Use no facilities that are unique to MVS/XA.
- Execute entirely in 31-bit addressing mode on an MVS/XA system and return control to its caller in 31-bit addressing mode. (The AMODE could be different from invocation to invocation.)
- Execute entirely in 24-bit addressing mode on an MVS/370 system.

The linkage editor and loader accept this combination from the ESD or CESD but not from the PARM field of the linkage editor EXEC statement or the linkage editor MODE control statement. The linkage editor converts AMODE ANY/RMODE ANY to AMODE 31/RMODE ANY.

Figure 5. AMODE and RMODE Combinations

Determining the AMODE and RMODE of a Load Module

Use the AMBLIST service aid to find out the AMODE and RMODE of a load module. The module summary produced by the LISTLOAD control statement contains the AMODE of the main entry point and the AMODE of each alias, as well as the RMODE specified for the load module. Refer to *Service Aids* for information about AMBLIST.

You can look at the source code to determine the AMODE and RMODE that the programmer intended for the program. However, the linkage editor or the loader can override these specifications.

Assembler H Support of AMODE and RMODE

Assembler H Version 2 supports AMODE and RMODE assembler instructions. Using AMODE and RMODE assembler instructions, you can specify an AMODE and an RMODE to be associated with a control section, an unnamed control section, or a named common control section. The assembler sets the AMODE and RMODE indicators in the ESD (external symbol dictionary) record for each control section specified.

AMODE and RMODE in the ESD

The assembler creates the external symbol dictionary, passing the information in it to the linkage editor or loader as part of the object module. A flags field (byte 12 of the ESD item in the ESD record) contains information about the addressing mode and residency mode of each CSECT. Bits 5, 6, and 7 of the flags field mean the following:

Bit 5:	0 - RMODE 24 1 - RMODE ANY
Bits 6-7:	00 - AMODE 24 (when the default is used) 01 - AMODE 24 (when AMODE 24 is specified) 10 - AMODE 31 11 - AMODE ANY

The assembler checks to determine if the specified AMODE/RMODE combination is valid. The only invalid combination is AMODE 24/ RMODE ANY.

The assembler also checks for the following error conditions:

- Multiple AMODE/RMODE statements for a single control section
- An AMODE/RMODE statement with an incorrect or missing value
- An AMODE/RMODE statement whose name field is not that of a valid control section in the assembly.

AMODE and RMODE Assembler Instructions

The AMODE instruction specifies the addressing mode to be associated with a CSECT in an object module. The format of the AMODE instruction is:

Name	Operation	Operand
Any symbol or blank	AMODE	24/31/ANY

The name field associates the addressing mode with a control section. If there is a symbol in the name field of an AMODE statement, that symbol must also appear in the name field of a START, CSECT, or COM statement in the assembly. If the name field is blank, there must be an unnamed control section in the assembly.

Similarly, the name field associates the residency mode with a control section. The RMODE statement specifies the residency mode to be associated with a control section. The format of the RMODE instruction is:

Name	Operation	Operand
Any symbol or blank	RMODE	24/ANY

Both the RMODE and AMODE instructions can appear anywhere in the assembly. Their appearance does not initiate an unnamed CSECT. There can be more than one RMODE (or AMODE) instruction per assembly, but they must have different name fields.

The defaults when AMODE, RMODE, or both are not specified are:

SPECIFIED	DEFAULTED
Neither	AMODE 24 RMODE 24
AMODE 24	RMODE 24
AMODE 31	RMODE 24
AMODE ANY	RMODE 24
RMODE 24	AMODE 24
RMODE ANY	AMODE 31

DFP Linkage Editor Support of AMODE and RMODE

The linkage editor accepts AMODE and RMODE specifications from any or all of the following:

- ESD (external symbol dictionary) entries in object modules.
- CESD (composite external symbol dictionary) entries in the load module.
- PARM field of the linkage editor EXEC statement. For example:

  ```
  //LKED EXEC PGM=name,PARM='AMODE=31,RMODE=ANY,.....'
  ```

  PARM field input overrides ESD and CESD input.

- Linkage editor MODE control statements in the SYSLIN data set. For example:

  ```
  MODE AMODE(31),RMODE(24)
  ```

  MODE control statement input overrides ESD, CESD, and PARM input.

Linkage editor processing results in two sets of AMODE and RMODE indicators located in:

- The CESD of the load module
- The PDS (partitioned data set) directory entry for the member name and any directory entries for alternate names or alternate entry points that were constructed using the linkage editor ALIAS control statement

These two sets of indicators may differ because they can be created from different input. The linkage editor creates indicators in the load module CESD based on input from the input ESD and CESD. The linkage editor creates indicators in the PDS directory based not only on input from the ESD and CESD, but also from the PARM field of the linkage editor EXEC statement; and the MODE control statements in the SYSLIN data set. The last two sources of input can override indicators from the ESD and CESD. Figure 6 shows linkage editor processing of AMODE and RMODE.

The linkage editor uses default values of AMODE 24/RMODE 24 for:

- Object modules produced by assemblers other than Assembler H Version 2
- Object modules produced by Assembler H Version 2 where source statements did not specify AMODE or RMODE
- Load modules produced by linkage editors other than the DFP linkage editor
- Load modules produced by the DFP linkage editor that did not have AMODE or RMODE specified from any input source
- Load modules in overlay structure

MVS/XA treats programs in overlay structure as AMODE 24, RMODE 24 programs. Putting a program into overlay structure destroys any AMODE and RMODE specifications contained in the CESD.

The linkage editor checks to see if the specified AMODE and RMODE combination is valid. The linkage editor recognizes as valid the following combinations of AMODE and RMODE:

AMODE 24 RMODE 24	
AMODE 31 RMODE 24	
AMODE 31 RMODE ANY	
AMODE ANY RMODE 24	
AMODE ANY RMODE ANY	Linkage editor accepts this combination from the ESD or CESD and places AMODE 31, RMODE ANY into the PDS directory entry (unless overridden by PARM values or MODE control statements). The linkage editor does not accept ANY/ANY from the PARM value or MODE control statement.

Any AMODE value specified alone in the PARM field or MODE control statement implies an RMODE of 24. Likewise, an RMODE of ANY specified alone implies an AMODE of 31. However, for RMODE 24 specified alone, the linkage editor does not assume an AMODE value. Instead, it uses the AMODE value specified in the ESD for the CSECT in generating the entry or entries in the PDS directory.

When the linkage editor creates an overlay structure, it assigns AMODE 24, RMODE 24 to the resulting program.

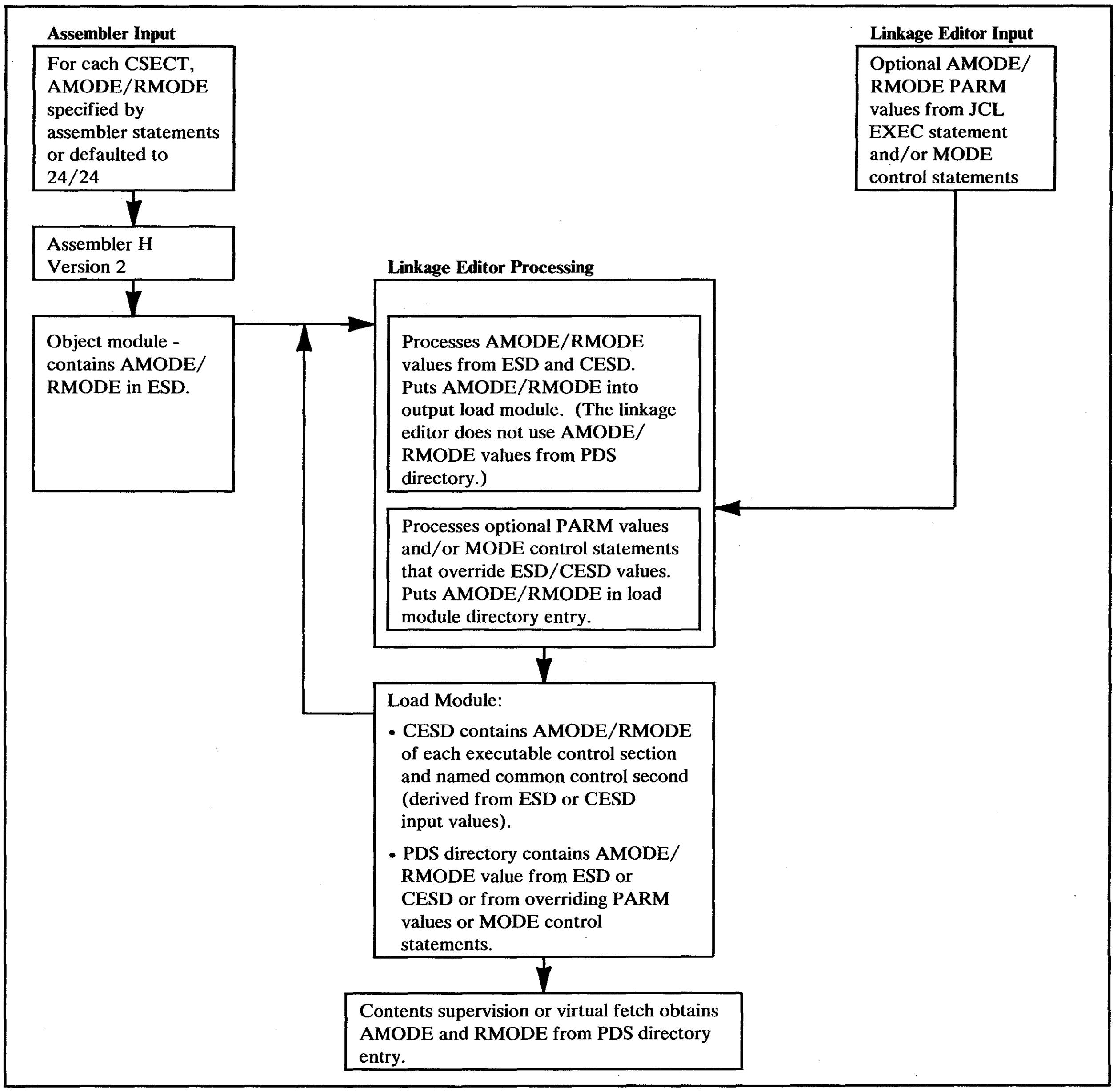

Assembler Input
For each CSECT, AMODE/RMODE specified by assembler statements or defaulted to 24/24
Assembler H Version 2
Object module - contains AMODE/ RMODE in ESD.
Linkage Editor Input
Optional AMODE/ RMODE PARM values from JCL EXEC statement and/or MODE control statements
Linkage Editor Processing
Processes AMODE/RMODE values from ESD and CESD. Puts AMODE/RMODE into output load module. (The linkage editor does not use AMODE/ RMODE values from PDS directory.)
Processes optional PARM values and/or MODE control statements that override ESD/CESD values. Puts AMODE/RMODE in load module directory entry.
Load Module:
• CESD contains AMODE/RMODE of each executable control section and named common control second (derived from ESD or CESD input values).
• PDS directory contains AMODE/ RMODE value from ESD or CESD or from overriding PARM values or MODE control statements.
Contents supervision or virtual fetch obtains AMODE and RMODE from PDS directory entry.

Figure 6. AMODE and RMODE Processing by the Linkage Editor

Linkage Editor RMODE Processing

In constructing a load module, the linkage editor frequently is requested to combine multiple CSECTs, or it may process an existing load module as input, combining it with additional CSECTs or performing a CSECT replacement.

The linkage editor determines the RMODE of each CSECT. If the RMODEs are all the same, the linkage editor assigns that RMODE to the load module. If the RMODEs are not the same (ignoring the RMODE specification on common sections), the more restrictive value, RMODE 24, is chosen as the load module's RMODE.

The RMODE chosen can be overridden by the RMODE specified in the PARM field of the linkage editor EXEC statement. Likewise, the PARM field RMODE can be overridden by the RMODE value specified on the linkage editor MODE control statement.

The linkage editor does not alter the RMODE values obtained from the ESD or CESD when constructing the new CESD. Any choice that the linkage editor makes or any override processing that it performs affects only the PDS directory entries.

DFP Loader Support for AMODE and RMODE

The loader's processing of AMODE and RMODE is similar to the linkage editor's. The loader accepts AMODE and RMODE specifications from:

ESDs in object modules
CESDs in load modules
PARM field of the JCL EXEC statement

Unlike the linkage editor, the loader does not accept MODE control statements from the SYSLIN data set, but it does base its loading sequence on the sequence of items in SYSLIN.

The loader passes the AMODE value to contents supervision. The loader processes the RMODE value as follows. If the user specifies an RMODE value in the PARM field, that value overrides any previous RMODE value. Using the value of the first RMODE it finds in the first ESD or CESD it encounters that is not for a common section, the loader obtains virtual storage for its output. As the loading process continues, the loader may encounter a more restrictive RMODE value. If, for example, the loader begins loading based on an RMODE ANY indicator and later finds an RMODE 24 indicator in a section other than a common section, it issues a message and starts over based on the more restrictive RMODE value. Figure 7 shows loader processing of AMODE and RMODE.

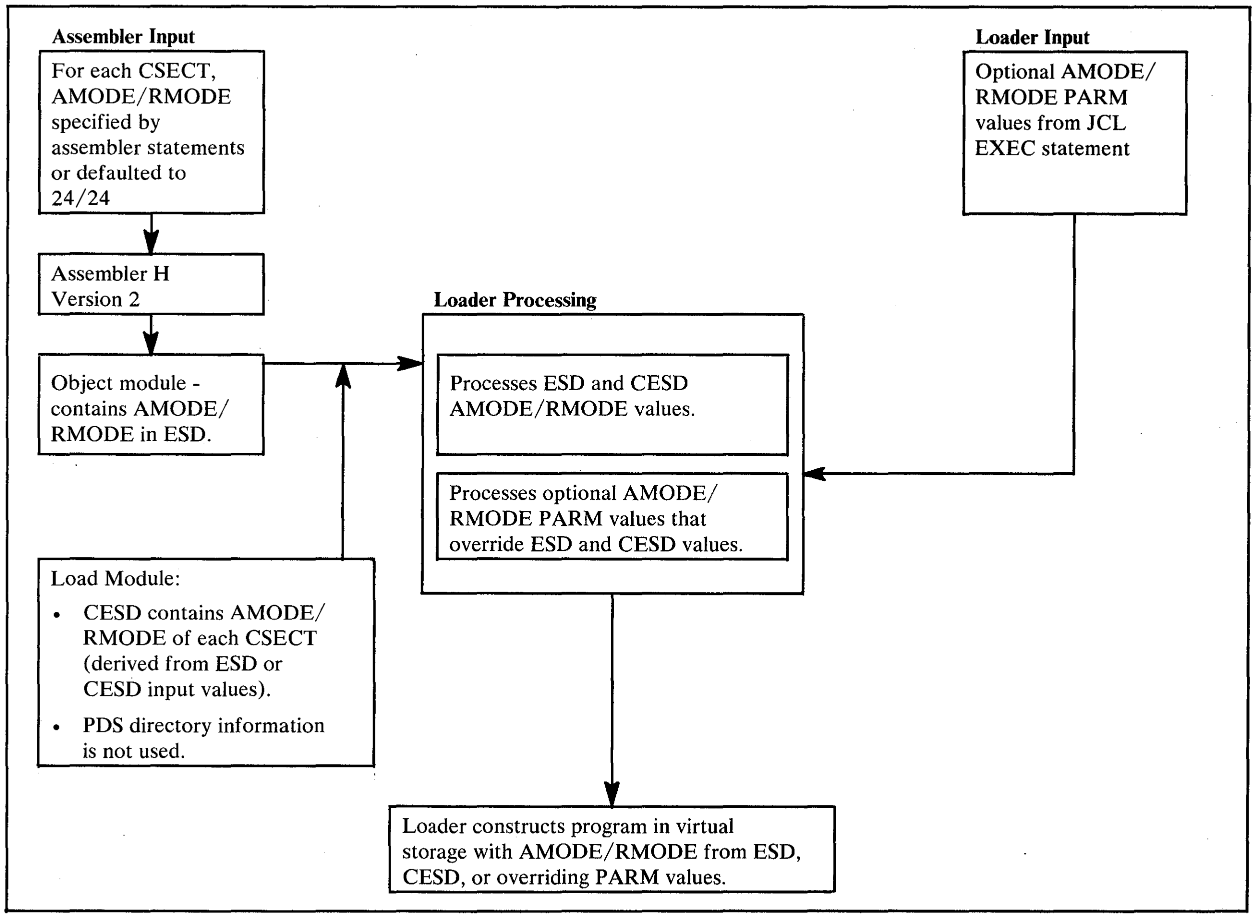

Assembler Input
For each CSECT, AMODE/RMODE specified by assembler statements or defaulted to 24/24
Assembler H Version 2
Object module - contains AMODE/ RMODE in ESD.
Load Module:
• CESD contains AMODE/ RMODE of each CSECT (derived from ESD or CESD input values).
• PDS directory information is not used.
Loader Processing
Processes ESD and CESD AMODE/RMODE values.
Processes optional AMODE/ RMODE PARM values that override ESD and CESD values.
Loader Input
Optional AMODE/ RMODE PARM values from JCL EXEC statement
Loader constructs program in virtual storage with AMODE/RMODE from ESD, CESD, or overriding PARM values.

Figure 7. AMODE and RMODE Processing by the Loader

MVS/XA's Support of AMODE and RMODE

The following are examples of MVS/XA's support of AMODE and RMODE:

- Program fetch obtains storage for the module as indicated by RMODE.
- ATTACH, LINK, and XCTL give the invoked module control in the addressing mode specified by its AMODE.
- LOAD brings a module into storage based on its RMODE and sets bit 0 in register 0 to indicate its AMODE.
- CALL passes control in the AMODE of the caller.
- SYNCH has an AMODE parameter that you can use to specify the AMODE of the invoked module.
- The SVC first level interrupt handler saves and sets the addressing mode.
- SRBs are dispatched in the addressing mode indicated by the SRB specified to the SCHEDULE macro.

- The cross memory instructions PC and PT establish the addressing mode for the target program.

- DFP access methods, except VSAM macros, support AMODE 24 RMODE 24 callers only. VSAM macros support all addressing and residency mode. callers.

- Dumping is based on the AMODE specified in the error-related PSW.

Program Fetch

Contents supervision places RMODE information from the PDS directory into the program fetch work area before calling program fetch. Program fetch issues the GETMAIN macro instruction to obtain storage above or below 16 megabytes as specified by the RMODE value.

ATTACH, LINK, and XCTL

Issuing an ATTACH macro instruction causes the control program to create a new task and indicates the entry point to be given control when the new task becomes active. If the entry point is a member name or an alias in the PDS directory, ATTACH gives it control in the addressing mode specified in the PDS directory entry or in the mode specified by the loader. If the invoked program has the AMODE ANY attribute, it gets control in the AMODE of its caller.

The LINK and XCTL macro instructions also give the invoked program control in the addressing mode indicated by its PDS directory entry for programs brought in by fetch or in the AMODE specified by the loader. The entry point specified must be a member name or an alias in the PDS directory passed by the loader, or specified in an IDENTIFY macro instruction. If the entry point is an entry name specified in an IDENTIFY macro instruction, IDENTIFY sets the addressing mode of the entry name equal to the addressing mode of the main entry point.

LOAD

Issuing the LOAD macro instruction causes the control program to bring the load module containing the specified entry point name into virtual storage (if a usable copy is not already there). LOAD sets the high-order bit of the entry point address in register 0 to indicate the module's AMODE (0 for 24, 1 for 31), which LOAD obtains from the module's PDS directory entry. If the module's AMODE is ANY, LOAD sets the high-order bit in register 0 to correspond to the caller's AMODE.

LOAD places the module in virtual storage either above or below the 16 megabyte line as indicated by the module's RMODE, which is specified in the PDS directory entry for the module.

Specifying the ADDR parameter indicates that you want the module loaded at a particular location. If you specify an address above 16 megabytes, be sure that the module being loaded has the RMODE ANY attribute. If you do not know the AMODE and RMODE attributes of the module, specify an address below 16 megabytes or omit the ADDR parameter.

CALL

The CALL macro instruction passes control to an entry point via BALR. Thus control is transferred in the AMODE of the caller. CALL does not change AMODE.

SYNCH

Using the AMODE parameter on the SYNCH macro instruction, you can specify the addressing mode in which the invoked module is to get control. Otherwise, SYNCH passes control in the caller's addressing mode.

SVC

For SVCs (supervisor calls), the supervisor saves and restores the issuer's addressing mode and makes sure that the invoked service gets control in the specified addressing mode.

SRB

When an SRB (service request block) is dispatched, MVS/XA sets the PSW A-mode bit based on the high-order bit of the SRBEP field. This bit, set by the issuer of the SCHEDULE macro, indicates the addressing mode of the routine operating under the dispatched SRB.

PC and PT

For a program call (PC), the entry table indicates the target program's addressing mode. The address field in the entry table must be initialized by setting the high-order bit to 0 for 24-bit addressing mode or to 1 for 31-bit addressing mode.

The PC instruction sets up register 14 with the return address and AMODE for use with the PT (program transfer) instruction. If PT is not preceded by a PC instruction, the PT issuer must set the high-order bit of the second operand register to indicate the AMODE of the program being entered (0 for 24-bit addressing mode or 1 for 31-bit addressing mode).

Data Management Access Methods

User programs must be in AMODE 24, RMODE 24 when invoking DFP access methods other than VSAM. All non-VSAM access methods require parameter lists, control blocks, buffers, and user exit routines to reside in virtual storage below 16 megabytes.

VSAM request macro instructions accept callers in AMODE 31, RMODE 24. Some macros allow parameter lists and control blocks to reside above 16 megabytes; for details on addressing and residence requirements for VSAM parameter lists, control blocks, buffers, and exit routines, see *VSAM Administration Guide.*

AMODE's Effect on Dumps

The only time AMODE has an effect on dumps is, for example, in a summary dump when data on either side of the address in each register is dumped. If the addresses in registers are treated as 24-bit addresses, the data dumped may come from a different storage location than when the addresses are treated as 31-bit addresses. If a dump occurs shortly after an addressing mode switch, some registers may contain 31-bit addresses and some may contain 24 bit addresses, but dumping services does not distinguish among them. Dumping services uses the AMODE from the error-related PSW. For example, in dumping the area related to the registers saved in the SDWA, dumping services uses the AMODE from the error PSW stored in the SDWA.

How to Change Addressing Mode

To change addressing mode you must change the value of the PSW A-mode bit. The following list includes all the ways to change addressing mode.

- The mode setting instructions BASSM and BSM.
- Supervisor-assisted linkage macro instructions (ATTACH, LINK, or XCTL). MVS/XA makes sure that routines get control in the specified addressing mode. Users need only ensure that parameter requirements are met. MVS/XA restores the invoker's mode on return from LINK.
- SVCs. The supervisor saves and restores the issuer's addressing mode and ensures that the service routine receives control in the addressing mode specified in its SVC table entry.
- SYNCH with the AMODE parameter to specify the addressing mode in which the invoked routine is to get control.
- An SRB. When the SRB is dispatched, MVS/XA sets the PSW A-mode bit with the high-order bit of the SRBEP field.
- The CIRB macro and the stage 2 exit effector. The CIRB macro is described in *SPL: System Macros and Facilities.*
- A PC or PT instruction. PC and PT instructions establish the specified addressing mode.
- An LPSW instruction (not recommended).

The example in Figure 8 illustrates how a change in addressing mode in a 24-bit addressing mode program enables the program to retrieve data from the OUXB control block, which might reside above 16 megabytes. The example works correctly whether or not the OUXB is actually above 16 megabytes. The example uses the BSM instruction to change addressing mode. In the example, the instruction L 2,4(,15) must be executed in 31-bit addressing mode. Mode setting code (BSM) before the instruction establishes 31-bit addressing mode and code following the instruction establishes 24-bit addressing mode.

```
USER     CSECT
USER     RMODE 24
USER     AMODE 24
         L   15,ASCBOUXB
         L   1,LABEL1    SET HIGH-ORDER BIT OF REGISTER 1 TO 1
                         AND PUT ADDRESS INTO BITS 1-31
         BSM 0,1         SET AMODE 31 (DOES NOT PRESERVE AMODE)
LABEL1   DC  A(LABEL2 + X'80000000')
LABEL2   DS  0H
         L   2,4(,15)    OBTAIN DATA FROM ABOVE 16 MEGABYTES
         LA  1,LABEL3    SET HIGH-ORDER BIT OF REGISTER 1 TO 0
                         AND PUT ADDRESS INTO BITS 1-31
         BSM 0,1         SET AMODE 24 (DOES NOT PRESERVE AMODE)
LABEL3   DS  0H
```

Figure 8. Mode Switching to Retrieve Data from Above 16 Megabytes

Chapter 4 - Establishing Linkage

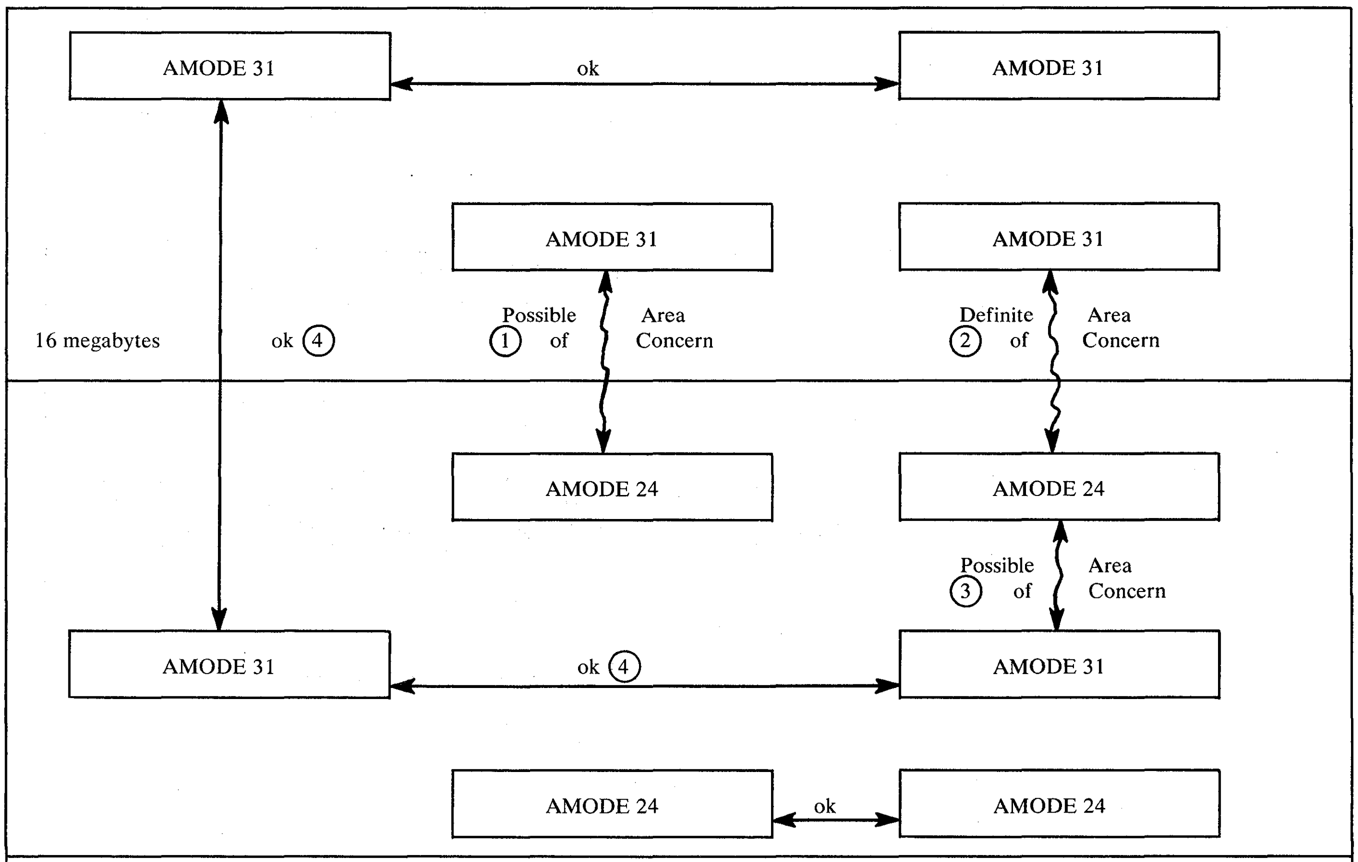
This chapter describes the mechanics of correct linkage in 31-bit addressing mode. Keep in mind that there are considerations other than linkage, such as locations of areas that both the calling module and the invoked module need to address.

Linkage in MVS/XA is the same as in MVS/370 for modules whose addressing modes are the same. As shown in Figure 9 , it is the linkage between modules whose addressing modes are different that is an area of concern. The areas of concern that appear in Figure 9 fall into two basic categories:

- Addresses passed as parameters from one routine to another must be addresses that both routines can use.
  - High-order bytes of addresses must contain zeroes or data that the receiving routine is programmed to expect.
  - Addresses must be less than 16 megabytes if they could be passed to a 24-bit addressing mode program.
- On transfers of control between programs with different AMODEs, the receiving routine must get control in the AMODE it needs and return control to the calling routine in the AMODE the calling routine needs.

There are a number of ways of dealing with the areas of concern that appear in Figure 9:

- Use the branching instructions (BASSM and BSM)
- Use pointer-defined linkage
- Use supervisor-assisted linkage (ATTACH, LINK, and XCTL)
- Use linkage assist routines
- Use "capping."



① When an AMODE 31 module that resides above the 16 megabyte line invokes an AMODE 24 module, the concerns are:

- The AMODE 24 program needs to receive control in 24-bit mode.
- The location of shared data (including control blocks, register save areas, and parameters). Can the AMODE 24 module address the data?
- The AMODE 24 module cannot return control unless an addressing mode change occurs.

② An AMODE 24 module cannot invoke an AMODE 31 module that resides above the line unless the AMODE 24 module changes its addressing mode either directly or using supervisor-assisted linkage.

③ When both modules are below 16 megabytes the concerns are:

- Which module cleans out bits 1-7 of the high-order bytes of 24-bit values used as addresses?
- Can both modules address shared data?

④ While there are no restrictions on the mechanics of linkage between two AMODE 31 modules, there might be restrictions on parameter values.

Figure 9. Linkage Between Modules with Different AMODEs and RMODEs

Using the BASSM and BSM Instructions

The BASSM (branch and save and set mode) and the BSM (branch and set mode) instructions are branching instructions that set the addressing mode. They are designed to complement each other. (BASSM is used to call and BSM is used to return, but they are not limited to such use.)

The description of BASSM from *Principles of Operation* appears in Figure 10.

BASSM R_1,R_2 [RR]

'OC'	R_1	R_2
0	8	12 15

Bits 32-63 of the current PSW, including the updated instruction address, are saved as link information in the general register designated by R_1. Subsequently, the addressing mode and instruction address in the current PSW are replaced from the second operand. The action associated with the second operand is not performed if the R_2 field is zero.

The contents of the general register designated by the R_2 field specify the new addressing mode and branch address; however when the R_2 field is zero, the operation is performed without branching and without setting the addressing mode.

When the contents of the general register designated by the R_2 field are used, bit 0 of the register specifies the new addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The new value for the PSW is computed before the register designated by R_1 is changed.

Condition Code: The code remains unchanged. **Program Exceptions:**
Trace (R_2 field is not zero).

Figure 10. BRANCH and SAVE and Set Mode Description

The description of BSM from *Principles of Operation* appears in Figure 11.

BSM R_1,R_2 [RR]

'0B'	R_1	R_2
0	8	12 15

Bit 32 of the current PSW, the addressing mode, is inserted into the first operand. Subsequently the addressing mode and instruction address in the current PSW are replaced from the second operand. The action associated with an operand is not performed if the associated R field is zero.

The value of bit 32 of the PSW is placed in bit position 0 of the general register designated by R_1, and bits 1-31 of the register remain unchanged; however, when the R_1 field is zero, the bit is not inserted, and the contents of general register 0 are not changed.

The contents of the general register designated by the R_2 field specify the new addressing mode and branch address; however, when the R_2 field is zero, the operation is performed without branching and without setting the addressing mode.

When the contents of the general register designated by the R_2 field are used, bit 0 of the register specifies the new addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The new value for the PSW is computed before the register designated by R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Figure 11. Branch and Set Mode Description

Calling and Returning with BASSM and BSM

In the following example, a module named BELOW has the attributes AMODE 24, RMODE 24. BELOW uses a LOAD macro to obtain the address of module ABOVE. The LOAD macro returns the address in register 0 with the addressing mode indicated in bit 0 (a pointer-defined value). BELOW stores this address in location EPABOVE. When BELOW is ready to branch to ABOVE, BELOW loads ABOVE's entry point address from EPABOVE into register 15 and branches using BASSM 14,15. BASSM places the address of the next instruction into register 14 and sets bit 0 in register 14 to 0 to correspond to BELOW's addressing mode. BASSM replaces the PSW A-mode bit with bit 0 of register 15 (a 1 in this example) and replaces the PSW instruction address with the branch address (bits 1-31 of register 15) causing the branch.

ABOVE uses a BSM 0,14 to return. BSM 0,14 does not save ABOVE's addressing mode because 0 is specified as the first operand register. It replaces the PSW A-mode bit with bit 0 of register 14 (BELOW's addressing mode set by BASSM) and branches.

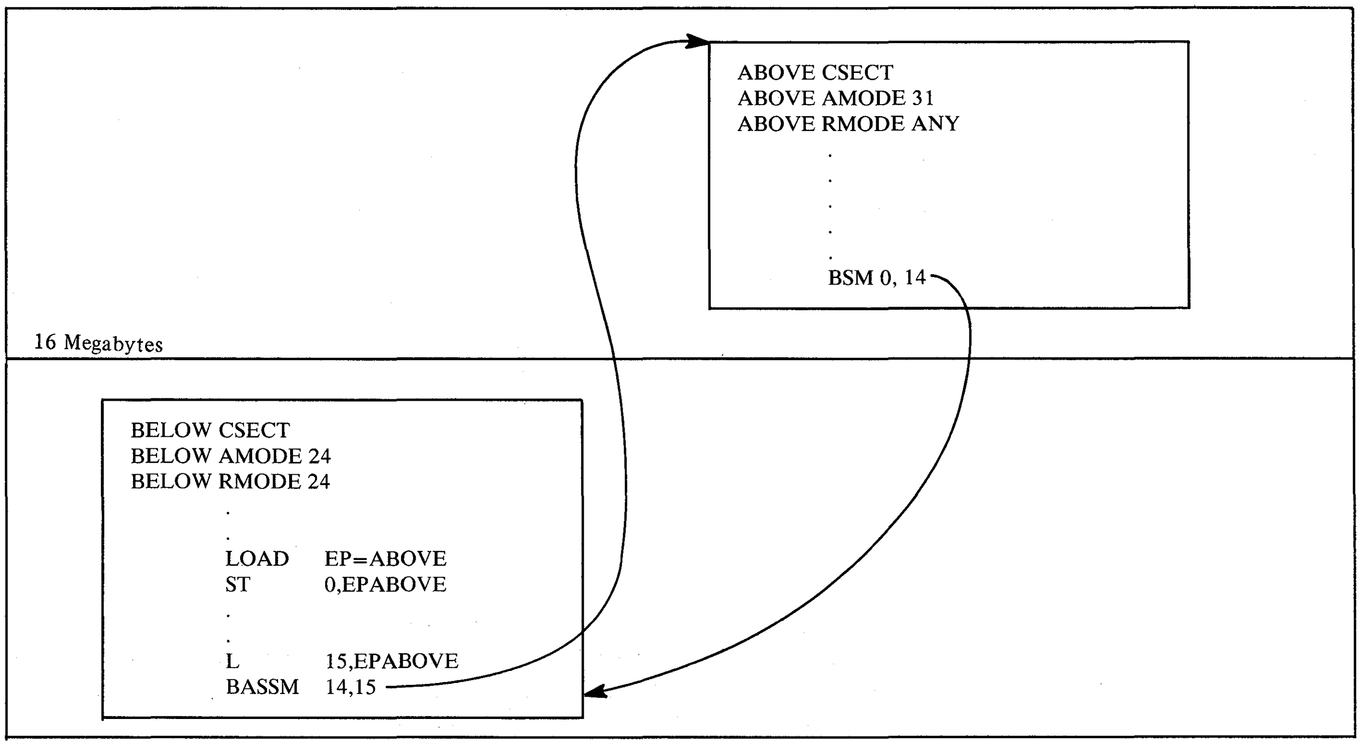


Figure 12. Using BASSM and BSM

Using Pointer-Defined Linkage

Pointer-defined linkage is a convention whereby programs can transfer control back and forth without having to know each other's AMODEs. Pointer-defined linkage is simple and efficient. You should use it in new or modified modules where there might be mode switching between modules.

Pointer-defined linkage uses a **pointer-defined value**, which is a 4-byte area that contains both an AMODE indicator and an address. The high-order bit contains the AMODE; the remainder of the word contains the address. To use pointer-defined linkage, you must:

- Use a pointer-defined value to indicate the entry point address and the entry point's AMODE. (The LOAD macro provides a pointer-defined value.)
- Use the BASSM instruction specifying a register that contains the pointer-defined value. BASSM saves the caller's AMODE and next the address of the next sequential instruction, sets the AMODE of the target routine, and branches to the specified location.
- Have the target routine save the full contents of the return register and use it in the BSM instruction to return to the caller.

Using an ADCON to Obtain a Pointer-Defined Value

The following method is useful when you need to construct pointer-defined values to use in pointer-defined linkages between control sections or modules that will be link edited into a single load module. You can also use this method when the executable program is prepared in storage using the loader.

The method requires the use of an externally-defined address constant in the routine to be invoked that identifies its entry mode and address. The address constant must contain a pointer-defined value. The calling program loads the pointer-defined value and uses it in a BASSM instruction. The invoked routine returns using a BSM instruction.

In Figure 13, RTN1 obtains pointer-defined values from RTN2 and RTN3. RTN1, the invoking routine does not have to know the addressing modes of RTN2 and RTN3. Later, RTN2 or RTN3 could be changed to use different addressing modes, and at that time their address constants would be changed to correspond to their new addressing mode. RTN1, however, would not have to change the sequence of code it uses to invoke RTN2 and RTN3.

You can use the techniques that the previous example illustrates to handle routines that have multiple entry points (possibly with different AMODE attributes). You need to construct a table of address constants, one for each entry point to be handled.

```
RTN1     CSECT
         EXTRN   RTN2AD
         EXTRN   RTN3AD
         .
         .
         L       15,=A(RTN2AD)    LOAD ADDRESS OF POINTER-DEFINED VALUE
         L       15,0(,15)        LOAD POINTER-DEFINED VALUE
         BASSM   14,15            GO TO RTN2 VIA BASSM
         .
         .
         L       15,=A(RTN3AD)    LOAD ADDRESS OF POINTER-DEFINED VALUE
         L       15,0(,15)        LOAD POINTER DEFINED-VALUE
         BASSM   14,15            GO TO RTN3 VIA BASSM
         .
RTN2     CSECT
RTN2     AMODE   24
         ENTRY   RTN2AD
         .
         BSM     0,14            RETURN TO CALLER IN CALLER'S MODE
RTN2AD   DC      A(RTN2)         WHEN USED AS A POINTER-DEFINED VALUE,
                                 INDICATES AMODE 24 BECAUSE BIT 0 IS 0
RTN3     CSECT
RTN3     AMODE   31
         ENTRY   RTN3AD
         .
         BSM     0,14                RETURN TO CALLER IN CALLER'S MODE
RTN3AD   DC      A(X'80000000'+RTN3) WHEN USED AS A POINTER-DEFINED VALUE
                                     INDICATES AMODE 31 BECAUSE BIT 0 IS 1
```

Figure 13. Example of Pointer-Defined Linkage

As with all forms of linkage, there are considerations over and above the linkage mechanism. These include:

- Both routines must have addressability to any parameters passed.
- Both routines must agree which of them will clean up any 24-bit addresses that might have extraneous information bits 1-7 of the high-order byte. (This is a consideration only for AMODE 31 programs.)

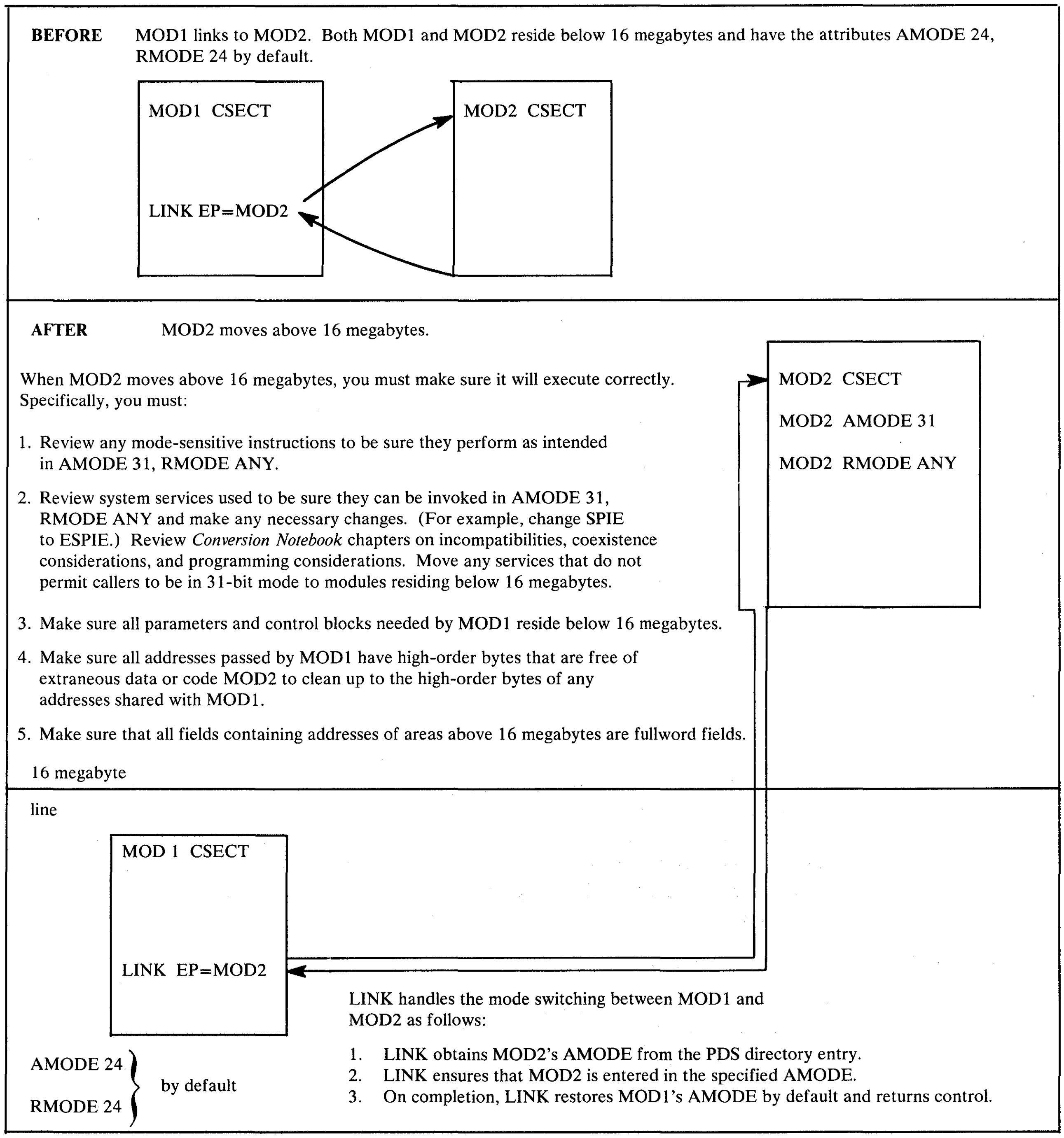
When a 24-bit addressing mode program invokes a module that is to execute in 31-bit addressing mode, the calling program must ensure that register 13 contains a valid 31-bit address of the register save area with no extraneous data in bits 1-7 of the high-order byte. In addition, when any program invokes a 24-bit addressing mode program, register 13 must point to a register save area located below 16 megabytes.

Using the LOAD Macro Instruction to Obtain a Pointer-Defined Value

LOAD returns a pointer-defined value in register 0. You can preserve this pointer-defined value and use it with a BASSM instruction to pass control without having to know the target routine's AMODE.

Using Supervisor-Assisted Linkage

Figure 14 shows a "before" and "after" situation involving two modules, MOD1 and MOD2. In the BEFORE part of the figure both modules execute in 24-bit addressing mode. MOD1 invokes MOD2 using the LINK macro instruction. The AFTER part of the figure shows MOD2 moving above 16 megabytes and outlines the steps that were necessary to make sure both modules continue to perform their previous function.



BEFORE MOD1 links to MOD2. Both MOD1 and MOD2 reside below 16 megabytes and have the attributes AMODE 24, RMODE 24 by default.

AFTER MOD2 moves above 16 megabytes.

When MOD2 moves above 16 megabytes, you must make sure it will execute correctly. Specifically, you must:

1. Review any mode-sensitive instructions to be sure they perform as intended in AMODE 31, RMODE ANY.
2. Review system services used to be sure they can be invoked in AMODE 31, RMODE ANY and make any necessary changes. (For example, change SPIE to ESPIE.) Review *Conversion Notebook* chapters on incompatibilities, coexistence considerations, and programming considerations. Move any services that do not permit callers to be in 31-bit mode to modules residing below 16 megabytes.
3. Make sure all parameters and control blocks needed by MOD1 reside below 16 megabytes.
4. Make sure all addresses passed by MOD1 have high-order bytes that are free of extraneous data or code MOD2 to clean up to the high-order bytes of any addresses shared with MOD1.
5. Make sure that all fields containing addresses of areas above 16 megabytes are fullword fields.

LINK handles the mode switching between MOD1 and MOD2 as follows:

1. LINK obtains MOD2's AMODE from the PDS directory entry.
2. LINK ensures that MOD2 is entered in the specified AMODE.
3. On completion, LINK restores MOD1's AMODE by default and returns control.

Figure 14. Example of Supervisor-Assisted Linkage

Linkage Assist Routines

A **linkage assist routine**, sometimes called an addressing mode interface routine, is a module that performs linkage for modules executing in different addressing or residency modes. Using a linkage assist routine, a 24-bit addressing mode module can invoke a 31-bit addressing mode module without having to make any changes. The invocation results in an entry to a linkage assist routine that resides below the 16 megabyte line and invokes the 31-bit addressing mode module in the specified addressing mode.

Conversely, a 31-bit addressing mode module, such as a new user module, can use a linkage assist routine to communicate with other user modules that execute in 24-bit addressing mode. The caller appears to be making a direct branch to the target module, but branches instead to a linkage assist routine that changes modes and performs the branch to the target routine.

The main advantage of using a linkage assist routine is to insulate a module from addressing mode changes that are occurring around it.

The main disadvantage of using a linkage assist routine is that it adds overhead to the interface. In addition, it takes time to develop and test the linkage assist routine. Some alternatives to using linkage assist routines are:

- Changing the modules to use pointer-defined linkage (described earlier in this chapter).
- Adding a prologue and epilogue to a module to handle entry and exit mode switching, as described later in this chapter under "Capping."

The MVS/XA control program uses several types of linkage assist routines. Among these are routines that get control in the addressing mode of the caller and perform the following operations:

- Save registers.
- Clean up parameter and linkage registers. If the values in these registers came from 24-bit addressing mode programs, bits 1-7 of the high-order bytes may contain unwanted flags and indicators.
- Obtain a new save area.
- Branch to the target routine via the BASSM instruction.
- Get control back from the target routine and restore the caller's addressing mode, if necessary. (The BSM instruction performs this function.)
- Restore the caller's registers.
- Return to the caller.

Example of Using a Linkage Assist Routine

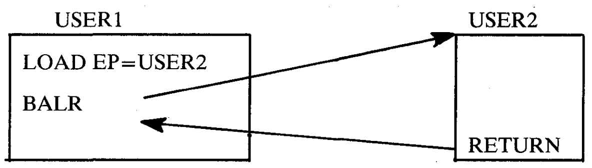
Figure 15 shows a "before" and "after" situation involving modules USER1 and USER2. USER1 invokes USER2 by using a LOAD and BALR sequence. The "before" part of the figure shows USER1 and USER2 residing below the 16 megabyte line and lists the changes necessary if USER2 moves above 16 megabytes. USER1 does not change.

The "after" part of the figure shows how things look after USER2 moves above 16 megabytes. Note that USER2 is now called USER3 and the newly created linkage assist routine has taken the name USER2.

The figure continues with a coding example that shows all three routines after the move.

BEFORE

Existing Application - USER1 invokes USER2 repeatedly



Change	Reason
• Change name of USER2 to USER3.	• USER1 does not have to change the LOAD USER2 macro instruction.
• Write a linkage assist routine called USER2.	• USER1 remains unchanged; new USER2 switches AMODEs and branches to USER3 (the former USER).
• Change USER3 (formerly USER2) as follows:	
- Make sure all control blocks and parameters needed by USER1 and USER2 are located below the 16 megabytes line.	- USER1 and USER2 are AMODE 24; they cannot access parameters or data above 16 megabytes.
- Check mode-sensitive instructions to be sure they perform the intended function in AMODE 31, RMODE ANY.	- USER3 was moved above 16 megabytes and has the attributes AMODE 31, RMODE ANY.
- Check system services used to be sure they can be invoked in AMODE 31, RMODE ANY and make any necessary changes. (For example, change SPIE to ESPIE.) Review *Conversion Notebook* chapters on incompatibilities coexistence, considerations, and programming considerations.	- USER3 has the attributes AMODE 31, RMODE ANY. SPIE and some other system services will not work in AMODE 31.
- Make sure that all fields containing addresses of areas above 16 megabytes are fullword fields.	

AFTER

Changed Application

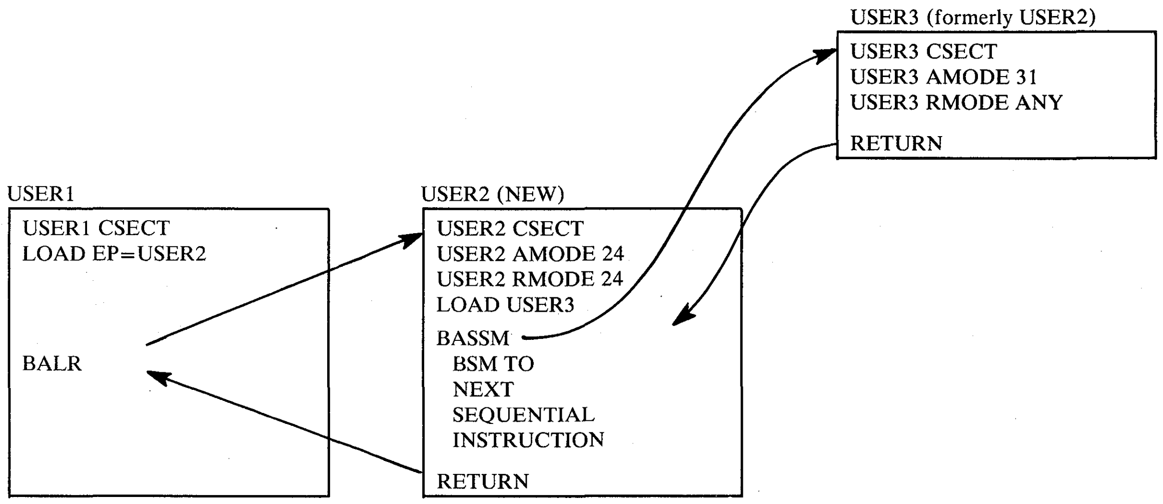


Figure 15 (Part 1 of 4). Example of a Linkage Assist Routine

USER1 (This module will not change)

```
*  USER MODULE USER1 CALLS MODULE USER2                                 00000100
USER1    CSECT                                                          00000200
BEGIN    SAVE   (14,12),,*  (SAVE REGISTER CONTENT, ETC.)               00000300
*  ESTABLISH BASE REGISTER(S) AND NEW SAVE AREA (NORMAL                 00000400
*  ENTRY CODING)                                                        00000500
           .
           .
*  ISSUE LOAD FOR MODULE USER2                                          00000700
         LOAD   EP=USER2    ISSUE LOAD FOR MODULE "USER2"               00000800
*  In the MVS/XA environment, the LOAD macro returns a
*  pointer-defined value.  However, because module USER1
*  has not been changed and executes in AMODE 24, the
*  the pointer-defined value has no effect on the BALR
*  instruction used to branch to module USER2.
         ST     0,EPUSER2   PRESERVE ENTRY POINT                        00000900
           .
*  MAIN PROCESS BEGINS                                                  00001000
PROCESS  DS     0H                                                      00001100
           .
           .
           .
           .
           .
           .
*  PREPARE TO GO TO MODULE USER2                                        00002000
         L      15,EPUSER2  LOAD ENTRY POINT                            00002100
         BALR   14,15                                                   00002200
           .
           .
           .
           .
         TM                 TEST FOR END                                00003000
         BC     PROCESS     CONTINUE IN LOOP                            00003100
           .
         DELETE EP=USER2
         L      13,4(13)
         RETURN (14,12),T,RC=0  MODULE USER1 COMPLETED                  00005000
EPUSER2  DC     F'0'        ADDRESS OF ENTRY POINT TO USER2             00007000
         END    BEGIN                                                   00007100
```

USER2 (Original application module)

```
*  USER MODULE USER2 (INVOKED FREQUENTLY FROM USER1)                    00000100
USER2    CSECT                                                          00000200
         SAVE   (14,12),,*  SAVE REGISTER CONTENT, ETC.                 00000300
*  ESTABLISH BASE REGISTER(S) AND NEW SAVE AREA (NORMAL                 00000400
*  ENTRY CODING)
           .
           .
           .
           .
           .
         L      13,4(13)
         RETURN (14,12),T,RC=0  MODULE USER2 COMPLETED                  00008100
         END                                                            00008200
```

Figure 15 (Part 2 of 4). Example of a Linkage Assist Routine

USER2 (New linkage assist routine)

```
* THIS IS A NEW LINKAGE ASSIST ROUTINE                              0000100
* (IT WAS NAMED USER2 SO THAT MODULE USER1 WOULD NOT                0000200
*  HAVE TO BE CHANGED)                                              0000300
USER2    CSECT                                                      0000400
USER2    AMODE  24                                                  0000500
USER2    RMODE  24                                                  0000600
         SAVE   (14,12),,*  (SAVE REGISTER CONTENT, ETC.)           0000700
*  ESTABLISH BASE REGISTER(S) AND NEW SAVE AREA (NORMAL             0000800
*  ENTRY CODING)
           .
* FIRST TIME LOGIC, PERFORMED ON INITIAL ENTRY ONLY,                0002000
* (AFTER INITIAL ENTRY, BRANCH TO PROCESS (SHOWN BELOW))            0002100
           .
         GETMAIN        NEW REGISTER SAVE AREA                      0003000
           .
         LOAD   EP=USER3                                            0004000
*  USER2 LOADS USER3 BUT DOES NOT DELETE IT.  USER2 CANNOT
*  DELETE USER3 BECAUSE USER2 DOES NOT KNOW WHICH OF ITS USES
*  OF USER3 IS THE LAST ONE.
         ST     0,EPUSER3   PRESERVE POINTER DEFINED VALUE          0004100
           .
* PROCESS  (PREPARE FOR ENTRY TO PROCESSING MODULE)                 0005000
           .
         (FOR EXAMPLE, VALIDITY CHECK REGISTER CONTENTS)
           .
           .
* PRESERVE AMODE FOR USE DURING RETURN SEQUENCE                     0007000
         LA     1,XRETURN            SET RETURN ADDRESS             0008000
         BSM    1,0                  PRESERVE CURRENT AMODE         0008100
         ST     1,XSAVE              PRESERVE ADDRESS               0008200
         L      15,EPUSER3           LOAD POINTER DEFINED VALUE     0009000
* GO TO MODULE USER3                                                0009100
         BASSM  14,15                TO PROCESSING MODULE           0009200
* RESTORE AMODE THAT WAS IN EFFECT                                  0009300
         L      1,XSAVE              LOAD POINTER DEFINED VALUE     0009400
         BSM    0,1                  SET ADDRESSING MODE            0009500
XRETURN  DS     0H                                                  0009600
         L      13,4(13)
           .
         RETURN (14,12),T,RC=0       MODULE USER2 HAS COMPLETED     0010000
EPUSER3  DC     F'0'                 POINTER DEFINED VALUE          0010100
XSAVE    DC     F'0'                 ORIGINAL AMODE AT ENTRY        0010200
         END                                                        0010500
```

- Statements 8000 through 8200: These instructions preserve the AMODE in effect at the time of entry into module USER2.
- Statement 9200: This use of the BASSM instruction:
  - Causes the USER3 module to be entered in the specified AMODE (AMODE 31 in this example). This occurs because the LOAD macro returns a pointer-defined value that contains the entry point of the loaded routine, and the specified AMODE of the module.
  - Puts a pointer-defined value for use as the return address into Register 14.
- Statement 9400: Module USER3 returns to this point.
- Statement 9500: Module USER2 re-establishes the AMODE that was in effect at the time the BASSM instruction was issued (STATEMENT 9200).

Figure 15 (Part 3 of 4). Example of a Linkage Assist Routine

USER3 (New Application Module)

```
*  MODULE USER3     (PERFORMS FUNCTIONS OF OLD MODULE USER2)        00000100
USER3    CSECT                                                      00000200
USER3    AMODE  31                                                  00000300
USER3    RMODE  ANY                                                 00000400
         SAVE   (14,12),,*  (SAVE REGISTER CONTENT, ETC.)           00000500
*  ESTABLISH BASE REGISTER(S) AND NEW SAVE AREA                     00000600
           .
           .
           .
           .
           .
           .
*  RESTORE REGISTERS AND RETURN                                     00008000
           .
         RETURN (14,12),T,RC=0                                      00008100
         END                                                        00008200
```

- Statements 300 and 400 establish the AMODE and RMODE values for this module. Unless they are over-ridden by linkage editor PARM values or MODE control statements, these are the values that will be placed in the PDS directory for this module.
- Statement 8100 returns to the invoking module.

Figure 15 (Part 4 of 4). Example of a Linkage Assist Routine

Using Capping - Linkage Using a Prologue and Epilogue

An alternative to linkage assist routines is a technique called **capping**. You can add a "cap" (prologue and epilogue) to a module to handle entry and exit addressing mode switching. The cap accepts control in either 24-bit or 31-bit addressing mode, saves the caller's registers, and switches to the addressing mode in which the module is designed to run. After the module has completed its function, the epilogue portion of the cap restores the caller's registers and addressing mode before returning control.

For example, when capping is used, a module in 24-bit addressing mode can be invoked by modules whose addressing mode is either 24-bit or 31-bit; it can perform its function in 24-bit addressing mode and can return to its caller in the caller's addressing mode. Capped modules must be able to accept callers in either addressing mode. Modules that reside above the 16 megabyte line cannot be invoked in 24-bit addressing mode. Capping, therefore, can be done only for programs that reside below the 16 megabyte line.

Some control program modules have addressing modes that are called CAP24 or CAP31. These addressing modes are not supported by the assembler, linkage editor, or loader. If you are debugging or modifying MVS/XA, you need to be aware of these terms and their definitions:

- **AMODE CAP24** - The control program module is designed to receive control from either 24-bit or 31-bit addressing mode callers, execute mainly in 24-bit addressing mode, and return control in the addressing mode of the caller.

- **AMODE CAP31** - The control program module is designed to receive control in either 24-bit or 31-bit addressing mode, execute mainly in 31-bit addressing mode, and return control in the addressing mode of the caller.

Figure 16 shows a cap for a 24-bit addressing mode module.

```
MYPROG    CSECT
MYPROG    AMODE ANY
MYPROG    RMODE 24
          USING *,15
          STM 14,12,12(13) SAVE CALLER'S REGISTERS BEFORE SETTING AMODE
          LA  10,SAVE      SET FORWARD ADDRESS POINTER IN CALLER'S
          ST  10,8(13)     SAVE AREA
          LA  12,MYMODE    SET AMODE BIT TO 0 AND ESTABLISH BASE
          LA  11,RESETM    GET ADDRESS OF EXIT CODE
          BSM 11,12        SAVE CALLER'S AMODE AND SET IT TO AMODE 24
          USING *,12
MYMODE    DS   0H
          DROP 15
          ST  13,SAVE+4    SAVE CALLER'S SAVE AREA
          LR  13,10        ESTABLISH OWN SAVE AREA

          This is the functional part of the original module.
          This example assumes that register 11 retains its
          original contents throughout this portion of the program.

          L   13,4(13)     GET ADDRESS OF CALLER'S SAVE AREA
          BSM 0,11         RESET CALLER'S AMODE
RESETM    DS  0H
          LM  14,12,12(13) RESTORE CALLER'S REGISTERS IN CALLER'S AMODE
          BR  14           RETURN
  SAVE    DS  0F
          DC  18F'0'
```

Figure 16. Cap for an AMODE 24 Module

Chapter 5 - Performing I/O in 31-Bit Addressing Mode

Programs in 31-bit addressing mode usually need to use 24-bit addressing mode programs to perform an I/O operation because all I/O control blocks, IDAWs (indirect data address words), and CCWs must reside below 16 megabytes and all I/O requests must be made by programs executing in 24-bit addressing mode (except for VSAM). Generally, data buffers must be below 16 megabytes as well.

A 31-bit addressing mode program can perform an I/O operation by:

- Using VSAM services that accept callers in either 24-bit or 31-bit addressing mode. Control blocks must also reside in virtual storage below 16 megabytes. (The *VSAM Administration Guide* describes VSAM services.)

- Using the EXCP macro instruction. All parameter lists, control blocks, CCWs, virtual IDAWs, and EXCP appendage routines must reside in virtual storage below 16 megabytes. A description of using EXCP to perform I/O appears later in this chapter.

- Using the EXCPVR macro instruction. All parameter lists, control blocks, CCWs, IDALs (indirect data address lists), and appendage routines must reside in virtual storage below 16 megabytes. A description of using EXCPVR to perform I/O appears later in this chapter.

- Invoking a routine that executes in 24-bit addressing mode as an interface to non-VSAM access methods, which accept callers executing in 24-bit addressing mode only. Chapter 4 described this method.

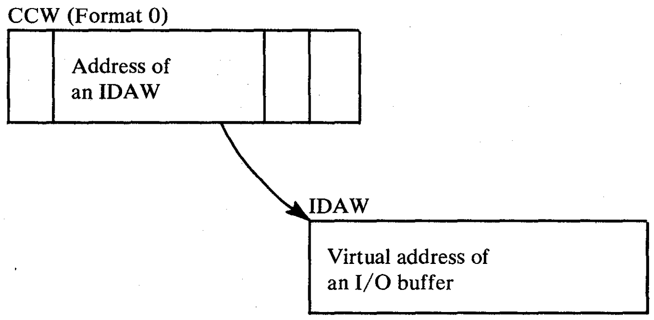
- Using the method shown in Figure 17 later in this chapter.

To perform I/O to buffers located in virtual storage above 16 megabytes, programs must use either:

- The EXCP macro instruction and virtual IDAWs.
- The EXCPVR macro instruction.

Using the EXCP Macro Instruction

EXCP macro instruction users can perform I/O to virtual storage areas above 16 megabytes. By using virtual IDAW support, CCWs in the EXCP channel program can, using a 24-bit address, point to a virtual IDAW that contains the 31-bit virtual address of an I/O buffer. The CCWs and IDAWs themselves must reside in virtual storage below 16 megabytes. The EXCP service routine supports only format 0 CCWs, the CCW format used in MVS/370.



Any CCW that causes data to be transferred can point to a virtual IDAW. Virtual IDAW support is limited to non-VIO data sets.

Although the I/O buffer can be in virtual storage above 16 megabytes, the virtual IDAW that contains the pointer to the I/O buffer and all the other areas related to the I/O operation (CCWs, IOBs, DEBs, and appendages) must reside in virtual storage below 16 megabytes.

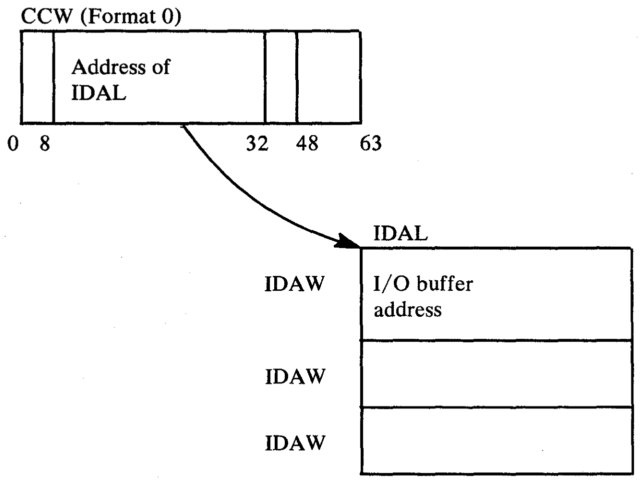
System-Data Administration describes how to use EXCP.

Using EXCPVR

Note: This topic concerns the use of real storage. Chapter 6 describes additional real storage considerations.

The EXCPVR interface supports only format 0 CCWs (the format used in MVS/370). Format 0 CCWs support only 24-bit addresses. All CCWs and IDAWs used with EXCPVR must reside in virtual or real storage below 16 megabytes. The largest virtual or real storage address you can specify directly in your channel program is 16 megabytes minus one. However, using IDAWs (indirect data address words) you can specify any real storage address and therefore you can perform I/O to any location in real or virtual storage. EXCPVR channel programs must use IDAWs to specify buffer addresses above 16 megabytes in real storage.

The format 0 CCW may contain the address of an IDAL (indirect address list), which is a list of IDAWs (indirect data address words).



You must assume that buffers obtained by data management access methods have real storage addresses above 16 megabytes. Data management access methods specify LOC=(BELOW,ANY) when they issue GETMAIN macro instructions. LOC=(BELOW,ANY) indicates that virtual storage can be backed with real storage above 16 megabytes when page fixed.

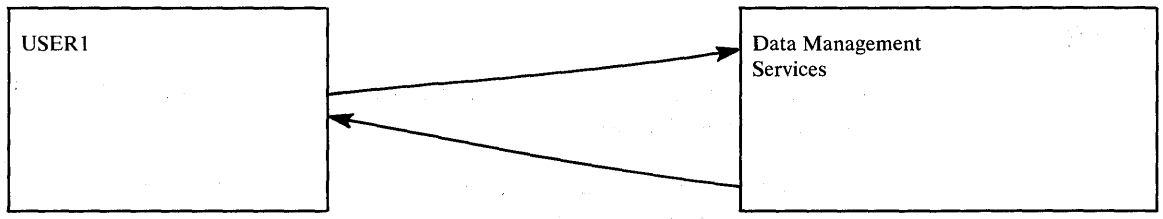
System-Data Administration describes how to use EXCPVR.

Example of Performing I/O While Residing Above 16 Megabytes

Figure 17 shows a "before" and "after" situation that involves two functions, USER1 and USER2. In the BEFORE part of the example, USER1 contains both functions and resides below 16 megabytes. In the AFTER part of the example USER1 has moved above the 16 megabyte line. The portion of USER1 that requests data management services has been removed and remains below 16 megabytes.

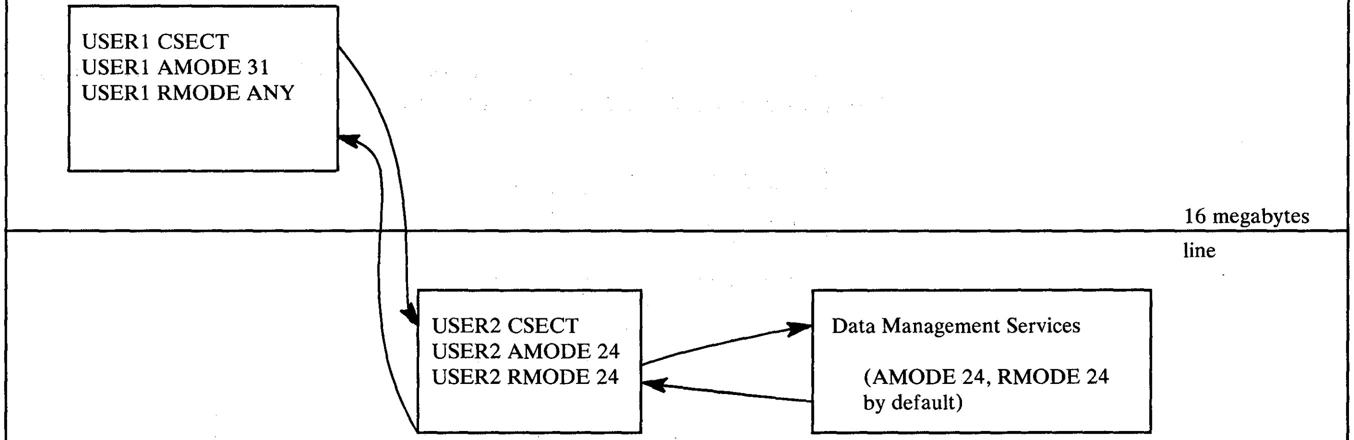
The figure includes a detailed coding example that shows both USER1 and USER2.

BEFORE



USER 1 is an application program that occasionally requests data management services to perform data base I/O. USER1 that requests services. USER1 and data management services reside below 16 megabytes.

AFTER



USER1 moves above the 16 megabyte line and moves its interface to data management into a new module, USER2. USER2 remains below the line because data management services must be invoked in 24-bit addressing mode (except for VSAM). The following coding example shows USER1 and USER2 after USER1 has moved.

Figure 17 (Part 1 of 13). Performing I/O While Residing Above 16 Megabytes

USER1 Application Module

```
*Module USER1 receives control in 31-bit addressing mode, resides in
*storage above 16 megabytes, and calls module USER2 to perform data
*management services.
*In this example, note that no linkage assist routine is needed.
USER1    CSECT
USER1    AMODE 31
USER1    RMODE ANY
*
*        Save the caller's registers in save area provided
*
#100     SAVE  (14,12)                 Save registers
         BASR  12,0                    Establish base
         USING *,12                    Addressability
```

Storage will be obtained via GETMAIN for USER2's work area (which will also contain the save area that module USER2 will store into as well as parameter areas in which information will be passed.) Since module USER2 must access data in the work area, the work area must be obtained below the 16 megabyte line.

```
         LA    0,WORKLNTH                   Length of the work area
*                                           required for USER2
#200     GETMAIN RU,LV=(0),LOC=BELOW        Obtain work area storage
         LR    6,1                          Save address of obtained
*                                           storage to be used for
*                                           a work area for module
*                                           USER2
         USING WORKAREA,6                   Work area addressability
```

The SAVE operation at statement **#100** may save registers into a save area that exists in storage either below or above the 16 megabyte line. If the save area supplied by the caller of module USER1 is in storage below the 16 megabyte line, it is assumed that that the high order byte of register 13 is zero.

The GETMAIN at statement **#200** must request storage below the 16 megabyte line for the following reasons:

1. The area obtained via GETMAIN will contain the register save area in which module USER2 will save registers. Because module USER2 runs in 24-bit addressing mode, it must be able to access the save area.
2. Because module USER2 will extract data from the work area to determine what function to perform, the area must be below the 16 megabyte line, otherwise, USER2 would be unable to access the parameter area.

Figure 17 (Part 2 of 13). Performing I/O While Residing Above 16 Megabytes

```
         LA     0,GMLNTH              Get dynamic storage for
*                                     module USER1 (USER1 resides
*                                     above the 16 megabyte line)
#300     GETMAIN RU,LV=(0),LOC=RES    Get storage above the 16
*                                     megabyte line
         LR     8,1                   Copy address of storage
*                                     obtained via GETMAIN
         USING DYNAREA,8              Base register for dynamic
*                                     work area
#400     ST     13,SAVEBKWD           Save address of caller's
*                                     save area
         LR     9,13                  Save caller's save area
*                                     address
         LA     13,SAVEAREA           USER1's save area address
*                                     Note - save area is below
*                                     the 16 megabyte line
         ST     13,8(9)               Have caller's save area
*                                     point to my save area
         LOAD   EP=IOSERV             Load address of data
*                                     management service
*                                     Entry point address
*                                     returned will be pointer-defined
         ST     0,EPA                 Save address of loaded
*                                     routine.
```

The GETMAIN at statement **#300** requests that the storage to be obtained match the current residency mode of module USER1. Because the module resides above the 16 megabyte line, the storage obtained will be above the 16 megabyte line.

At statement **#400**, the address of the caller's save area is saved in storage below the 16 megabyte line.

Figure 17 (Part 3 of 13). Performing I/O While Residing Above 16 Megabytes

```
Prepare to open input and output data base files

         MVC    FUNCTION,OPEN1          Indicate open file 1
*                                       for input
         LA     1,COMMAREA              Set up register 1 to
*                                       point to the parameter
*                                       area
#500     L      15,EPA                  Get pointer-defined address
*                                       of the I/O service
*                                       routine
#600     BASSM  14,15                   Call the service routine
*                                       Note: AMODE will change
#650     MVC    FUNCTION,OPEN2          Indicate open file 2
*                                       for output
         LA     1,COMMAREA              Setup register 1 to
*                                       point to the parameter
*                                       area
#700     L      15,EPA                  Get pointer-defined address
*                                       of the I/O service
*                                       routine
         BASSM  14,15                   Call the service routine
*                                       Note: AMODE will change
```

The entry point address loaded at statements **#500** and **#700** is pointer-defined, as returned by the LOAD service routine. That is, the low-order three bytes of the symbolic field EPA will contain the virtual address of the loaded routine while the high order bit (bit 0) will be zero to indicate the loaded module is to receive control in 24-bit addressing mode. The remaining bits (1-7) will also be zero in the symbolic field EPA.

The BASSM at statement **#600** does the following:

- Places into bit positions 1-31 of register 14 the address of statement **#650**.
- Sets the high-order bit of register 14 to one to indicate the current addressing mode.
- Replaces bit positions 32-63 of the current PSW with the contents of register 15 (explained above)

The BSM instruction used by the called service routine USER2 to return to USER1 will reestablish 31-bit addressing mode.

Figure 17 (Part 4 of 13). Performing I/O While Residing Above 16 Megabytes

Prepare to read a record from data base file 1.

```
READRTN  DS     0H
         MVC    FUNCTION,READ1           Indicate read to file 1
         XC     BUFFER,BUFFER            Clear input buffer
         LA     1,COMMAREA               Set up register 1 to
*                                        point to the parameter
*                                        area
         L      15,EPA                   Get pointer-defined address
*                                        of the I/O service
*                                        routine
         BASSM  14,15                    Call the service routine
*                                        Note: The BASSM will change
*                                        the AMODE to 24-bit.  The
*                                        BSM issued in the service
*                                        routine will reestablish
*                                        31-bit addressing mode.
#900     CLC    STATUS,ENDFILE           End of file encountered
*                                        by module USER2 ?
         BE     EODRTN                   Close files and exit
         MVC    BUFFR31A,BUFFER          Move record returned to
*                                        storage above the 16 megabyte
*                                        line
```

At statement **#900**, a check is made to determine if called module USER2 encountered the end of file. The end of file condition in this example can only be intercepted by USER2 because the EOD exit address specified on the DCB macro must reside below the 16 megabyte line. The end of file condition must then be communicated to module USER1.

Figure 17 (Part 5 of 13). Performing I/O While Residing Above 16 Megabytes

Call a record analysis routine that exists above the 16 megabyte line.

```
         LA     1,BUFFR31A               Get address of first buffer
         ST     1,BUFPTR+0               Store into parameter list
         LA     1,UPDATBFR               Get address of output
*                                        buffer
         ST     1,BUFPTR+4               Store into parameter list
         LA     1,BUFPTR                 Set up pointers to work
*                                        buffers for the analysis
*                                        routine
         L      15,ANALYZE               Address of analysis routine
#1000    BASR   14,15                    Call analysis routine
         MVC    BUFFER,UPDATBFR          Move updated record to
*                                        storage below the 16 megabyte
*                                        line so that the
*                                        updated record can
*                                        be written back to the
*                                        data base
```

At statement **#1000** a **BASR** instruction is used to call the analysis routine since no AMODE switching is required. A **BALR** could also have been used. A **BALR** executed while in 31-bit addressing mode performs the same function as a **BASR** instruction. The topic "Mode Sensitive Instructions" in Chapter 1 describes the instruction differences.

Figure 17 (Part 6 of 13). Performing I/O While Residing Above 16 Megabytes

```
         MVC   FUNCTION,WRITE1          Indicate write to file 1
         LA    1,COMMAREA               Set up register 1 to
*                                       point to the parameter
*                                       area
         L     15,EPA                   Get pointer-defined address
*                                       of the I/O service
*                                       routine
         BASSM 14,15                    Call the service routine
*                                       Note: The BASSM will set
*                                       the AMODE to 24-bit.  The
*                                       BSM issued in the service
*                                       routine will reestablish
*                                       31-bit addressing mode
         B     READRTN                  Get next record to
*                                       process
```

Prepare to close input and output data base files

```
EODRTN   DS    0H                       End of data routine
         MVC   FUNCTION,CLOSE1          Indicate close file 1
         LA    1,COMMAREA               Set up register 1 to
*                                       point to the parameter
*                                       area
         L     15,EPA                   Get pointer-defined address
*                                       of the I/O service
*                                       routine
         BASSM 14,15                    Call the service routine
*                                       Note: The BASSM sets
*                                       the AMODE to 24-bit.  The
*                                       BSM issued in the service
*                                       routine will reestablish
*                                       31-bit addressing mode
         MVC   FUNCTION,CLOSE2          Indicate close file 2
         LA    1,COMMAREA               Set up register 1 to
*                                       point to the parameter
*                                       area
         L     15,EPA                   Get pointer-defined address
*                                       of the I/O service
*                                       routine
         BASSM 14,15                    Call the service routine
*                                       Note: The BASSM sets
*                                       the AMODE to 24-bit.  The
*                                       BSM issued in the service
*                                       routine will reestablish
*                                       31-bit addressing mode
```

Figure 17 (Part 7 of 13). Performing I/O While Residing Above 16 Megabytes

Prepare to return to the caller

```
         L      13,SAVEBKWD               Restore save area address
*                                         of the caller of module
*                                         USER1
         LA     0,WORKLNTH                Length of work area and
*                                         parameter area used by
*                                         module USER2
         FREEMAIN RC,LV=(0),A=(6)         Free storage
         DROP  6
         LA     0,GMLNTH                  Length of work area used
*                                         by USER1
         FREEMAIN RC,LV=(0),A=(8)         Free storage
         DROP  8
         XR     15,15                     Set return code zero
         RETURN (14,12),RC=(15)
```

Define DSECTs and constants for module to module communication

Define constants used to communicate the function module USER2 is to perform.

```
         DS     0F
READ1    DC     C'R1'                     Read from file 1 opcode
WRITE1   DC     C'W1'                     Write to file 1 opcode
OPEN1    DC     C'O1'                     Open file 1 opcode
OPEN2    DC     C'O2'                     Open file 2 opcode
CLOSE1   DC     C'C1'                     Close file 1 opcode
CLOSE2   DC     C'C2'                     Close file 2 opcode
ANALYZE  DC     V(ANALYSIS)               Address of record
*                                         analysis routine
ENDFILE  DC     C'EF'                     End of file indicator



WORKAREA DSECT
SAVEREGS DS     0F                        This storage exists
*                                         below the 16 megabyte line
SAVEAREA EQU    SAVEREGS
SAVERSVD DS     F                         Reserved
SAVEBKWD DS     F
SAVEFRWD DS     F
SAVE1412 DS     15F                       Save area for registers 14-12
COMMAREA DS     0F                        Parameter area used to
*                                         communicate with module
*                                         USER2
FUNCTION DS     CL2                       Function to be performed
*                                         by USER2
STATUS   DS     CL2                       Status of read operation
BUFFER   DS     CL80                      Input/output buffer
WORKLNTH EQU    *-WORKAREA                Length of this DSECT
```

Figure 17 (Part 8 of 13). Performing I/O While Residing Above 16 Megabytes

Define DSECT work area for module USER1

```
DYNAREA   DSECT                            This storage exists
*                                          above the 16 megabyte line
EPA       DS     F                         Address of loaded routine
BUFFR31A  DS     CL80                      Buffer - above the 16 megabyte
*                                          line
BUFPTR    DS     0F
          DS     A                         Address of input buffer
          DS     A                         Address of updated buffer
UPDATBFR  DS     CL80                      Updated buffer returned
*                                          by the analysis routine
GMLNTH    EQU    *-DYNAREA                 Length of dynamic area
          END
```

Figure 17 (Part 9 of 13). Performing I/O While Residing Above 16 Megabytes

USER2 Service Routine

```
*Module USER2 receives control in 24-bit addressing mode, resides below
*the 16 megabyte line, and is called by module USER1 to perform data
*management services.

USER2    CSECT
USER2    AMODE 24
USER2    RMODE 24
*
*        Save the caller's registers in save area provided
*
         SAVE  (14,12)                  Save registers
         BASR  12,0                     Establish base
         USING *,12                     Addressability
         LR    10,1                     Save parameter area pointer
*                                       around GETMAINs
         USING  COMMAREA,10             Establish parameter area
*                                       addressability
```

Storage will be obtained via GETMAIN for a save area that module USER2 can pass to external routines it calls.

```
         LA    0,WORKLNTH               Length of the work area
*                                       required
         GETMAIN RU,LV=(0),LOC=RES      Obtain save area storage,
*                                       which must be below the
*                                       16 megabyte line
         LR    6,1                      Save address of obtained
*                                       storage to be used for
*                                       a save area for module
*                                       USER2
         USING SAVEREGS,6               Save area addressability
         LA    0,GMLNTH                 Get dynamic storage for
*                                       module USER2 below the
*                                       16 megabyte line.
```

Note: This GETMAIN will only be done on the initial call to this module.

```
#2000    GETMAIN RU,LV=(0),LOC=RES      Obtain storage below
*                                       the 16 megabyte line
         LR    8,1                      Copy address of storage
*                                       obtained via GETMAIN
         USING DYNAREA,8                Base register for the
*                                       dynamic work area
         ST    13,SAVEBKWD              Save address of caller's
*                                       save area
         LR    9,13                     Save caller's save area
*                                       address
         LA    13,SAVEAREA              USER1's save  area address
*                                       Note - save area is
*                                       below the 16 megabyte line
```

The GETMAIN at statement **#2000** requests that storage be obtained to match the current residency mode of module USER2. Because the module resides below the 16 megabyte line, storage obtained will be below the 16 megabyte line.

Figure 17 (Part 10 of 13). Performing I/O While Residing Above 16 Megabytes

Note: The following store operation is successful because module USER1 obtained save area storage below the 16 megabyte line.

```
         ST    13,8(9)                  Have caller's save area
*                                       point to my save area
         .
         .
         .
         .
* Process input requests
         .
         .
         .
         .
READ1    DS    0H                       Read a record routine
         .
         L     13,SAVEBKWD
         LM    14,12,12(13)             Reload USER1's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
WRITE1   DS    0H                       Write a record routine
         .
         L     13,SAVEBKWD
         LM    14,12,12(13)             Reload USER1's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
OPEN1    DS    0H                       Open file 1 for input
         .
         L     13,SAVEBKWD
         LM    14,12,12(13)             Restore caller's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
CLOSE1   DS    0H                       Close file 1 for input
         .
         L     13,SAVEBKWD
         LM    14,12,12(13)             Restore caller's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
OPEN2    DS    0H                       Open file 2 for input
         .
         L     13,SAVEBKWD
         LM    14,12,12(13)             Restore caller's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
CLOSE2   DS    0H                       Close file 2 for input
         .
         L     13.SAVEBKWD
         LM    14,12,12(13)             Restore caller's registers
         BSM   0,14                     Return to caller - this
*                                       instruction sets AMODE 31
         .
         .
```

Figure 17 (Part 11 of 13). Performing I/O While Residing Above 16 Megabytes

Note: This FREEMAIN will only be done on the final call to this module.

```
         LA     0,GMLNTH                  Length of work area used
*                                         by USER2
         FREEMAIN RC,LV=(0),A=(8)         Free storage
         .
         .
         .
```

DCB END OF FILE and ERROR ANALYSIS ROUTINES

```
ERREXIT1 DS     0H                        End of file encountered
         .
         .
         MVC    STATUS,ENDFILE            Indicate end of file to
*                                         module USER1
         .
         L      13,SAVWBKWD
         LM     14,12,12(13)              Reload USER1's registers
         BSM    0,14                      Return to USER1
*                                         indicating end of file
*                                         has been encountered
         .
         .
         .
         .
ERREXIT2 DS     0H                        SYNAD error exit
         .
         .
         MVC    STATUS,IOERROR            Indicate I/O error to
*                                         module 'USER1'
         .
         L      13,SAVWBKWD
         LM     14,12,12(13)              Reload USER1's registers
         BSM    0,14                      Return to USER1
*                                         indicating an I/O error
*                                         has been encountered
```

Figure 17 (Part 12 of 13). Performing I/O While Residing Above 16 Megabytes

Note: Define the required DCBs that module USER2 will process. The DCBs exist in storage below the 16 megabyte line. The END OF DATA and SYNAD exit routines also exist in storage below the 16 megabyte line.

```
INDCB    DCB   DDNAME=INPUT1,DSORG=PS,MACRF=(GL),EODAD=ENDFILE, x
               SYNAD=ERREXIT1


OUTDCB   DCB  DDNAME=OUTPUT1,DSORG=PS,MACRF=(PL),SYNAD=ERREXIT2


* Work areas and constants for module USER2

IOERROR  DC    C'IO'                   Constant used to indicate
*                                      an I/O error
ENDFILE  DC    C'EF'                   Constant used to indicate
*                                      end of file encountered


SAVEREGS DSECT                         This storage exists
*                                      below the 16 megabyte line
SAVEAREA EQU   SAVEREGS
SAVERSVD DS    F                       Reserved
SAVEBKWD DS    F
SAVEFRWD DS    F
SAVE1412 DS    15F                     Save area for registers 14-12
WORKLNTH EQU   *-SAVEREGS              Length of dynamic area
         .
         .
         .
         .
         .
COMMAREA DSECT                         Parameter area used to
*                                      communicate with module
*                                      USER1
FUNCTION DS    CL2                     Function to be performed
*                                      by USER2
STATUS   DS    CL2                     Status of read operation
BUFFER   DS    CL80                    Input/output buffer
         .
         .
DYNAREA  DSECT                         This storage exists
*                                      below the 16 megabyte line
         .
         .
         .
         .
         .
         .
GMLNTH   EQU   *-DYNAREA               Length of dynamic area
         .
         .
         END
```

Figure 17 (Part 13 of 13). Performing I/O While Residing Above 16 Megabytes

Chapter 6 - Understanding the Use of Real Storage

MVS/XA programs and data reside in virtual storage that, when necessary, is backed by real storage. Most programs and data do not depend on their real storage addresses. Some MVS/XA programs, however, do depend on real addresses and some require these real addresses to be less than 16 megabytes. MVS/XA reserves as much real storage below 16 megabytes as it can for such programs and, for the most part, handles their real storage dependencies without requiring them to make any changes.

The system uses the area of real storage above 16 megabytes to back virtual storage with real frames whenever it can. All virtual areas above 16 megabytes can be backed with real frames above 16 megabytes. In addition, the following virtual areas below 16 megabytes can also be backed with real frames above 16 megabytes:

- SQA
- LSQA
- Nucleus
- Pageable private areas
- Pageable CSA
- PLPA
- MLPA
- Resident BLDL

The following virtual areas are always backed with real frames below 16 megabytes:

- V=R regions
- FLPA
- Subpool 226
- Subpools 227 and 228 (unless specified otherwise by the GETMAIN macro instruction with the LOC parameter)

When satisfying page-fix requests, MVS/XA backs pageable virtual pages that reside below 16 megabytes with real storage below 16 megabytes, unless the GETMAIN macro specifies LOC=(BELOW,ANY) or the PGSER macro specifies the ANYWHER parameter. If the GETMAIN macro specifies or implies a real location of ANY, MVS/XA backs pageable virtual pages with real frames above 16 megabytes even when the area is page fixed. MVS/XA ignores requests to fix storage in the nucleus, SQA, or LSQA because those areas are already fixed. Real addresses in those areas might, therefore, be greater than 16 megabytes.

Real Storage Considerations for User Programs

Among the things to think about when dealing with real storage in 31-bit addressing mode are the use of the load real address (LRA) instruction, the use of the LOC parameter on the GETMAIN macro instruction, the location of the DAT-off portion of the nucleus, and using EXCPVR to perform I/O. (EXCPVR was described in Chapter 5.)

Load Real Address (LRA) Instruction

The LRA instruction always results in a 31-bit real address regardless of the issuing program's addressing mode. All programs that issue an LRA instruction must be prepared to handle a 31-bit result if the virtual storage address specified could have been backed with real storage above 16 megabytes. Issue LRA only against areas that are fixed.

GETMAIN Macro Instruction

The LOC parameter on the RU, RC, VRU, and VRC forms of the GETMAIN macro instruction specifies not only the virtual storage location of the area to be obtained, but also the real storage location when the storage is page fixed.

LOC=BELOW indicates that the virtual storage is to be located below 16 megabytes. When the area is page fixed, it is to be backed with real storage below 16 megabytes.

LOC=(BELOW,ANY) indicates that virtual storage is to be located below 16 megabytes but that real storage can be located either above or below 16 megabytes.

LOC=ANY and LOC=(ANY,ANY) indicate that both virtual and real storage can be located either above or below 16 megabytes.

LOC=RES indicates that the location of virtual and real storage depends on the location (RMODE) of the GETMAIN issuer. If the issuer has an RMODE of 24, LOC=RES indicates the same thing as LOC=BELOW; if the issuer has an RMODE of ANY, LOC=RES indicates the same thing as LOC=ANY.

LOC=(RES,ANY) indicates that the location of virtual storage depends on the location of the issuer but that real storage can be located anywhere.

LOC is optional except in the following case: A program that resides above the 16 megabyte line and uses the RU, RC, VRU, and VRC forms of GETMAIN **must** specify either LOC=BELOW or LOC=(BELOW,ANY) on GETMAIN requests for storage that will be used by programs running in 24-bit addressing mode. Otherwise, virtual storage would be assigned above 16 megabytes and 24-bit addressing mode programs could not use it.

The location of virtual storage can also be specified on the PGSER (page services) macro instruction. The ANYWHER parameter on PGSER specifies that a particular virtual storage area can be placed either above or below 16 megabytes on future page-ins. This parameter applies to virtual storage areas where LOC=(BELOW,ANY) or LOC=(ANY,ANY) was not specified on GETMAIN.

DAT-Off Routines

The MVS/370 nucleus is mapped so that its virtual storage addresses are equal to its real storage addresses. MVS/370 has a V=R (virtual=real) nucleus. In contrast, The MVS/XA nucleus is mapped and fixed in real storage without attempting to make its virtual storage addresses equal to its real storage addresses. MVS/XA has a V=F (virtual=fixed) nucleus.

Because the MVS/370 is V=R, nucleus code can turn DAT off, and the next instruction executed is the same as it would be if DAT was still on. Because the

MVS/XA nucleus is not V=R, MVS/XA nucleus code cannot turn DAT-off and expect the next instruction executed to be the same as if DAT was on.

To allow for the execution of DAT-off nucleus code, the MVS/XA nucleus consists of two load modules, one that runs with DAT on and one that runs with DAT off. Nucleus code that needs to run with DAT off must reside in the DAT-off portion of the nucleus.

When the system is initialized, the DAT-off portion of the nucleus is loaded into the highest contiguous real storage. Therefore, you must modify any user modules in the nucleus that run with DAT off so that they operate correctly above the 16 megabyte line. Among the things you may have to consider are:

- All modules in the DAT-off portion of the nucleus have the attributes AMODE 31, RMODE ANY. They may reside above 16 megabytes.

- These modules must return control via a BSM 0,14.

- Register 0 must not be destroyed on return.

To support modules in the DAT-off nucleus:

- Move the DAT-off code to a separate module with AMODE 31, RMODE ANY attributes. Use as its entry point, IEAVEURn where n is a number from 1 to 4. (MVS/XA reserves four entry points in the DAT-off nucleus for users.) Use BSM 0,14 as the return instruction. Do not destroy register 0.

- Code a DATOFF macro instruction to invoke the DAT-off module:

  ```
      DATOFF INDEX=INDUSRn
  ```

  The value of n in INDUSRn must be the same as the value of n in IEAVEURn, the DAT-off module's entry point.

- Link edit the DAT-off module (IEAVEURn) into the IEAVEDAT member of SYS1.NUCLEUS (the DAT-off nucleus).

Refer to *SPL: System Modifications* and *SPL: System Macros and Facilities* for more information about modifying the DAT-off portion of the nucleus and the DATOFF macro instruction.

Index

GC28-1158-1

MVS/Extended Architecture 31-Bit Addressing (File No. S370-36) Printed in U.S.A. GC28-1158-1