SLAC-PUB-5 187
March, 1990
(M)

EXPLOITING VM/XA*

Chuck Boeheim

Stanford Linear Accelerator Center
Stanford University, Stanford, CA 94309

ABSTRACT

The Stanford Linear Accelerator Center has recently completed a conversion to IBM's VM/XA SP Release 2 operating system. The primary physics application had been constrained by the previous 16 megabyte memory limit. Work is underway to enable this application to exploit the new features of VM/XA.

This paper presents a brief tutorial on how to convert an application to exploit VM/XA and discusses some of the SLAC experiences in doing so.

* Work supported by Department of Energy contract DE-AC03-76SF00515

Presented at SHARE 74,
Anaheim, CA, March 4-9, 1990

Search for the Z^0

Physicists at the Stanford Linear Accelerator Center are attempting to measure the mass of the Z^0 particle, one of the particles that mediate the weak nuclear force. This project has seen the conversion of the existing two-mile-long linear accelerator into a unique linear collider. Electrons and positrons are accelerated to near the velocity of light, separated and steered around two opposing arcs to meet head-on in beams six microns in cross-section. Some of these particles collide, and some of these collisions produce the massive Z^0 particle, which in turn decays into a number of other particles. The Z^0 exists for such a short time that it can be detected only by detecting the particles into which it decomposes.

Each of these "events" is painstakingly recorded by a detector, which will log about 256K bytes per event. In addition, Monte-Carlo simulations of events will log 400K bytes per event. Researchers hope to log around one hundred thousand of each type of event per year. The total data storage needs of this project will be around one terrabyte of data per year.

A complex data-analysis system has been developed for the processing of these events. At around 200,000 lines of MORTRAN, a FORTRAN preprocessor language, it's a hefty 6 megabytes of object code. It provides for batch data reduction and interactive analysis, including physics simulation, detector simulation, event reconstruction, event display (3D visualization), interactive data analysis (IDA), and a data manager. An additional 4 megabytes of descriptive constants are required, plus at least one megabyte of data for a typical analysis.

Under VM/SP about 10.5 megabytes were available beneath the shared segments for CMS and other programs required for interactive work. The complete program simply couldn't fit in memory, forcing its use in individual segments. On the VAX/VMS system where this software also ran, workspaces of 40 megabytes were often used. A solution was needed to provide for equivalent address spaces on the IBM equipment where this program did most of its production work.

What is XA?

The term XA is used in at least two different ways. It is

- A new machine architecture that allows the use of more than 16 megabytes of memory, up to two gigabytes, and also provides a more efficient and flexible I/O system. We'll refer to this as ***XA architecture.'***
- A version of the VM operating system that takes advantage of the new machine architecture. We'll refer to this as *VM/XA*.

A time of change

There are in general two kinds of changes needed when converting a program to VM/XA:

- Changes for XA architecture: memory addresses and virtual device addresses are larger, and there is a completely different set of I/O instructions. Some instructions available in XA architecture are not available in 370 architecture and vice versa.
- Changes for XA support: many CMS services required changes to deal with XA architecture. CP commands and their responses changed. There were large changes in the services for loading and generating modules-and for memory management.

SLAC took the opportunity of this conversion to clean up a decade's accumulation of utilities on the system. Utilities with duplicate or overlapping function were eliminated. A few more were eliminated because they were too difficult to convert to XA. A good deal of the change apparent to the users was because of this cleanup.

For the purposes of XA conversion, programs may fall into one of three categories: 370-only, XA toleration, and XA exploitation. 370-only ("compatibility mode") programs may execute only in a 370 architecture virtual machine. Generally they execute some instruction that is valid only in 370 architecture. A partial list of these instructions is given later in this paper. You may still need to make some program changes to achieve this level of compatibility because of changed responses from CP and CMS commands, changes to the rules for program loading and storage management, and parallel changes to shared segment sizes, locations, and usages.

XA toleration programs may execute in either a 370 or an XA architecture machine. They do not take advantage of XA features, such as the expanded address space. An example of a familiar

1. IBM manuals often refer to ***XA*** mode and 370 *mode*. Since so many other terms in VM refer to modes, we'll use the more precise term architecture.

program that is only XA tolerant, at least in VM/XA SP Release 2, is XEDIT. Even if you have 100 megabytes of virtual storage, you cannot edit any file that will not fit in the available memory below 16 megabytes. You would need to make the same conversions for these programs as for 370-only programs, plus changes because of 370 instructions that are not valid in XA architecture, because the XA PSW has a different format, or because interrupts are handled differently. These changes would most often be found in assembler language programs or subroutines.

XA exploitation programs take advantage of new XA features, such as larger address spaces. It is possible to write programs that run only in XA architecture and not in 370, but you would rarely find such a program in practice. You would need to worry about the previously-mentioned conversions, in addition the program must handle 31-bit addresses and must use some new CMS services. The last section of this paper will describe how to do that.

The good news is that most of the high-level languages such as FORTRAN, PL/I, and C create object code that is in this third category. If you write entirely in these languages, your programs will be all ready for XA exploitation.

Conversion Strategy

There are many strategies for conversion, and they have been covered well in other SHARE talks, so I'll just briefly describe the method that SLAC chose. We acquired a used 3083 CPU for the duration of the conversion effort (about 12 months). We brought up VM/XA SP Release 2 on. the 3083, while the 3090/3081 complex continued to run HPO 4.2 and CMS 5.0. The minidisk-sharing component of ISF was installed in HPO, and the CSE SPE was installed in VM/XA. This enabled the systems to share all minidisks with protection from multiple write links.

Early in the conversion the 3083 ran only an abbreviated directory, allowing only staff members to log on. After the majority of the system was in place, the full directory was loaded. Users were able, and encouraged, to log on to their accounts on the 3083 to test software.

When converting a program for XA, you can choose either to make a new version of the program that executes only in VM/XA, or to make a version that executes either in the old system or the new. The strategy that you choose will depend on the individual program and the way that it is supplied to the user. If there is only one common disk, you will need to make the program dual-

path; it will need to decide at execution time which system it is running under and which services it can use. If you have separate disks for each system, you can simply make a new, VM/XA-only version. You will need to evaluate this for each program that needs conversion. A few programs may need a major redesign for XA, and it may not be practical to dual-path them.

Subroutine libraries can pose a special problem. You may convert the subroutines in the libraries, but unless the users relink their programs with the new library, they will not get the benefit of the change. We found that we not only had to convert the subroutines, but also make them available on the HPO system as well as the XA system well in advance of the conversion date, to give users an opportunity to relink their programs. All subroutines had to be dual-path. We did not want users to have to relink their programs when they logged on to the XA system for testing and then to relink again when they logged back on the the HPO system. We also did not want to force everyone to relink on the morning of the final conversion.

We supplied different Y disks on the two systems, both to accommodate the few program products that had VM/XA-only versions, and to avoid massive perturbations of the production system as we obtained and installed new VM/XA-compatible versions of programs.

Working with VM/XA

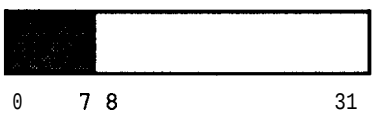
When your system is running VM/XA, the real machine is using the XA architecture. However, each virtual machine may choose independently to use either 370 or XA architecture. The SET MACHINE command chooses an architecture (see Figure 1). You must re-IPL CMS after a SET MACHINE command, just as you do after DEFINE STORAGE.

```
query set
MSG ON , WNG ON , EMSG TEXT, ACNT OFF, RUN OFF
LINEDIT ON , TIMER ON , ISAM OFF, ECMODE ON
ASSIST OFF , PAGEX OFF, AUTOPOLL OFF
IMSG ON , SMSG OFF AFFINITY NONE , NOTRAN OFF
VMSAVE OFF, 370E OFF'
STBYPASS OFF , STMULTI OFF 00/000
MIH OFF , VMCONIO OFF , CPCONIO OFF , SVCACCL OFF, CONCEAL OFF
MACHINE 370, SVC76 CP, NOPDATA OFF, IOASSIST OFF
CCWTRAN ON
Ready; T=0.01/0.01 11:02:24

set machine xa
System reset.
System = XA

ipl cms
SLAC XA CMS 205 10/03/89 14:45
```

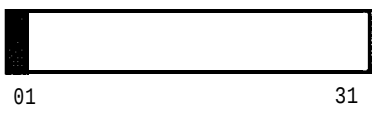
Figure I: Selecting an architecture

Addresses: the long and short of it

In a 370 architecture machine, a memory address was the low-order 24 bits of a fullword. In XA architecture you have a choice: it can be either the low-order 24 bits or the low-order 31 bits (see Figure 2). Bit zero is never used in an address.



370 Architecture ***or*** XA Architecture, AMODE 24



XA Architecture, AMODE 31

Figure 2: Addressing Modes

The choice is called the ***Addressing Mode*** or *AMODE*. AMODE 24 is for compatibility: an address is only 24 bits, and the program can address only memory less than 16 megabytes (2^{24} bytes). The first 8 bits of an address word are ignored. AMODE 31 uses 31 bits for an address,

which allows you to address up to 2 gigabytes of memory (2^{31} bytes).

Which type of address the program is using is determined by the AMODE bit in the PSW. When your program is started by CMS, it will be started in the AMODE that it requested. There are assembler instructions to change AMODE as needed. These are mainly needed when transferring control between programs that need different AMODEs, or when working with data areas or parameter lists passed between unlike programs.

The AMODE attribute of a program designates which AMODE it can execute in. AMODE 24 means that the program is not capable of handling 31 bit addresses and must receive control in AMODE 24. AMODE 31 means that the program is capable of handling 31 bit addresses and must receive control in AMODE 31. AMODE ANY means that the program is capable of handling either 31 or 24 bit addresses, and receives control in the AMODE of the calling program.

If you're having trouble making sense of the preceding paragraph, notice that we're using AMODE in two different ways. The AMODE attribute of a program tells CMS the addressing mode to use when starting a program. The AMODE of an executing program is determined by a bit in the PSW, and the program can change it as needed. The AMODE attribute of a program is simply a way to get your program started off in the correct AMODE.

The AMODE attribute is assigned to a TEXT file by the compiler. Most current compilers such as VS FORTRAN and PL/I assign AMODE ANY to their TEXT files. Some older compilers assign AMODE 24. In assembler, the AMODE statement can select any AMODE for the object file. The AMODE of a MODULE defaults to the AMODE of the first TEXT file included in it.

In VM/XA your virtual machine is divided into two areas: the memory less than or equal to sixteen megabytes, and any memory greater than that. The memory below 16 megabytes is that addressable in AMODE 24; all memory is addressable in AMODE 31. This division is so important that it's called the ***sixteen megabyte line.***

If a program is AMODE 24, it must execute below the 16 megabyte line; if it is AMODE 31 or ANY, it may execute either above or below the line. CMS itself runs just below the line, but has some of its data areas just above the line (see Figure 3).

Where the program executes is called its ***Residency Mode*** or ***RMODE.*** There are only two choices: RMODE 24 means the program must execute below 16 megabytes. The program may have any AMODE. RMODE ANY means the program may execute either above or below the line. If sufficient memory is available above the line, it will load there. If not, it will load below. A -program with RMODE ANY cannot have AMODE 24.

The RMODE of a MODULE defaults to ANY if all of its components' AMODEs are 31 or ANY. RMODE is forced to 24 if any included TEXT has AMODE 24.

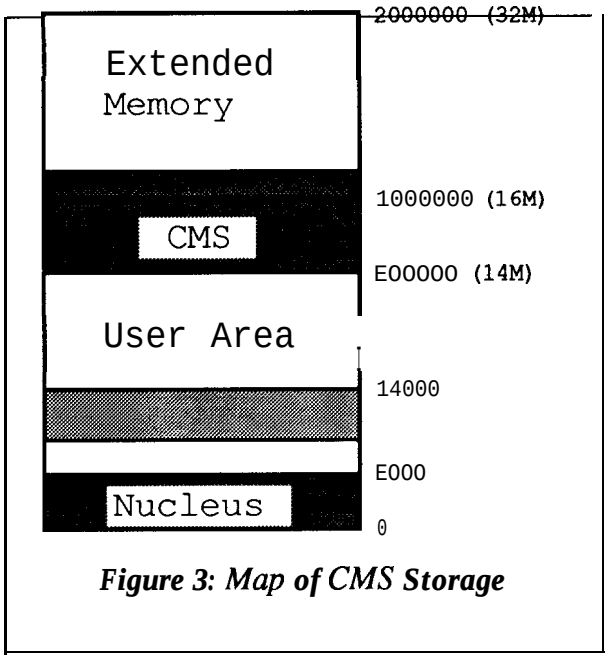


Figure 3: Map of CMS Storage

It can be puzzling at times to determine the AMODEs and RMODEs of TEXT and MODULE files. IBM provides no easy way to determine these attributes, so I wrote two commands, TEXTINFO and MODINFO, to help out (see Figure 4, page 7). These two commands are available on VMSHARE or from SLAC.

```
textinfo test
Name      Length    Amode Rmode
TEST      000000B8  Any   Any
Ready; T=0.01/0.01 14:50:46

textinfo fortslac txtlib
Name      Length    Amode Rmode
A2INT     0000009E    24    24
BLKEST    00000000  Any   Any
CHK218    00000000  Any   Any
CONVER7   00000110    24    24
...
Ready; T=0.02/0.06 10:35:30

modinfo mount
Name      Origin    Length    Entry Pt Amode Rmode Arch Attributes
MOUNT     00020000  00010BB0  00024AC8  24   24    Any  Noclean...
Ready; T=0.01/0.01 10:41:05
```

Figure 4: TEXTINFO and MODINFO commands

Loading Programs

Where a TEXT file is loaded by CMS is determined by the architecture of the virtual machine, the RMODE of the first TEXT file, the SET LOADAREA command, and the ORIGIN option of the LOAD command. All of these interact in a complex fashion; see the table on page 46 of CMS Application Program Conversion Guide (SC23-0403).

RMODE ANY programs are loaded at the beginning of the largest free area above the 16 megabyte line, or below the line if no such storage is available. RMODE 24 programs are loaded at the beginning of the largest free area under the line (usually X'14000'). You can use the ORIGIN option to specify a particular address, but it may conflict with other options, and you must make sure that the memory is not already in use.

The new SET LOADAREA command specifies the default area to use. Somewhat confusingly, the default for this command is different for 370 and XA architecture virtual machines. SET LOADAREA RESPECT loads TEXT files according to their AMODE and is the default in an XA virtual machine. SET LOADAREA 20000 loads text files at address X'20000' and is the default in a 370 virtual machine.

The new PROGMAP command is very handy for finding out where your program loaded (see Figure 5). It's useful for determining the interaction of the various LOAD defaults and options, and required for finding addresses to use with the TRACE (PER) command.

```
load test31
Ready;  T=0.01/0.10 11:57:44
progmap
Name       Entry      Origin     Bytes        Attributes
TEST31     01021000   01021000   00000008     Amode 31   Non-reloc
Ready;  T=0.01/0.01 11:57:47
load test24
Ready;  T=0.01/0.08 11:57:58
progmap
Name       Entry      Origin     Bytes        Attributes
TEST24     00014000   00014000   00000008     Amode 24   Non-reloc
Ready;  T=0.01/0.01 11:58:01
```

Figure 5: The PROGMAP Command

If you are loading more than one TEXT file, and a subsequent TEXT file has a more restrictive AMODE than the first, the load will restart below the line. This is almost never what you want. If you change the disk search order between the LOAD command and a subsequent INCLUDE the LOAD would be unable to restart. If you have this situation you should try either to fix the more restrictive AMODE or to specify AMODE 24 explicitly on the LOAD command so that it starts correctly (see Figure 6).

```
load test31
Ready; T=0.02/0.18 11:58:23
progmap
Name      Entry     Origin    Bytes        Attributes
TEST31    01021000  01021000  00000008    Amode 31  Non-reloc
Ready; T=0.01/0.01 12:08:00
include test24
Restrictive RMODE encountered in CSECT TEST24.
LOAD continues below 16Mb.
Ready; T=0.01/0.09 12:08:10
progmap
Name      Entry     Origin    Bytes        Attributes
TEST24    00014008  00014008  00000008    Amode 24  Non-reloc
TEST31    00014000  00014000  00000008    Amode 31  Non-reloc
Ready: T=0.01/0.01 11:58:27
```

Figure 6: Restarting a LOAD

A corollary is that a linked program must go either entirely above or entirely below the 16 megabyte line. If you must do this, e.g., for a program that won't fit below the line but must use some subroutines that are AMODE 24, you will have to split the program. You can dynamically load subroutines from TXTLIBs or LOADLIBs and LINK to them, though transferring control in the correct AMODE can be tricky. Dynamically acquired data areas can also be split from the program; these include FORTRAN dynamic commons.

Generating Modules

There are two kinds of modules in VM/XA: relocatable and non-relocatable. Relocatable modules load in memory at the highest address appropriate to their RMODE. Non-relocatable modules load at the fixed address at which they were generated. Non-relocatable modules are the more traditional type of CMS module, and they may be the simplest to use when initially converting programs. However, relocatable modules are very important for RMODE ANY modules. If you make an RMODE ANY non-relocatable module in a small or 370 virtual machine, it

will never run above the line, because it always runs at the fixed address below the line. If you make it in a large XA machine, it will never run in a smaller virtual machine.

You may override the AMODE and RMODE attributes on both the LOAD and GENMOD commands. Just because you can doesn't mean you should, however. You can safely downgrade AMODE and RMODE of ANY or 31 to 24, because that is more restrictive. It's generally not safe to upgrade AMODE or RMODE 24 to ANY or 31. It is possible to have TEXT files marked AMODE and RMODE 24 that you know to be 31-bit capable. However, it's always worth taking the time to recompile or reassemble the routine to get it marked correctly.

A backwards compatibility note: You can generate modules in VM/XA that execute properly in previous releases of CMS if you watch out for a few "gotchas". The extra information about AMODE and RMODE etc. is generally ignored by earlier CMSs. However, the meaning and default of the STR/NOSTR option of GENMOD has changed. The result is that a program that uses OS storage macros (including FORTRAN, PL/I, et.al.) that is linked and GENMODed in VM/XA will not work under earlier CMSs if you use the default option. If you explicitly specify the STR option on the GENMOD in VM/XA, the resulting module will execute properly in all CMS releases.

You also have to be careful of the default of SET LOADAREA RESPECT in an XA architecture virtual machine. This will cause modules to be generated at X'14000'. Such a module cannot load properly in earlier releases of CMS. Use SET LOADAREA 20000 or an explicit ORIGIN option to maintain backwards compatibility.

Overlays -- Don't do it!

VM/XA now manages programs in memory. Older versions of CMS did not attempt to do this and allowed programs to be overlaid. CMS 5.5 now deletes previously loaded programs that occupy the memory needed by subsequent programs. Programs that depend on the old behavior to produce defacto overlay structures will no longer work.

LOADing a TEXT file also deletes previously loaded programs, unless the PRES option is used. However, even if two programs are simultaneously loaded, they remain two independent entities. In Figure 7, two programs are loaded at non-overlapping addresses. A subsequent PROGMAP command shows that the first program has disappeared. They are loaded again, this time with the PRES option on the second one. This time both programs remain in memory. However, the

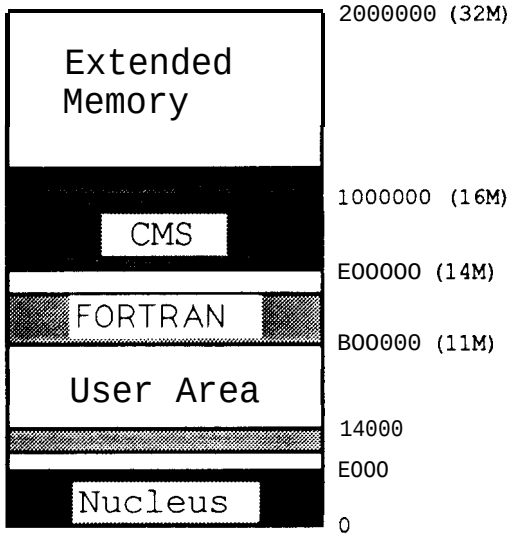
```
load test24
Ready;  T=0.01/0.08 16:32:16

load test31 (origin 15000
Ready;  T=0.01/0.02 16:32:29

progmap
Name       Entry      Origin     Bytes        Attributes
TEST31     00015000   00015000   00000008    Amode 31   Non-reloc
Ready;  T=0.01/0.01 16:32:32

load test24
Ready;  T=0.01/0.08 16:32:39

load test31 (origin 15000 pres
Ready;  T=0.01/0.02 16:32:47

progmap
Name       Entry      Origin     Bytes        Attributes
TEST31     00015000   00015000   00000008    Amode 31   Non-reloc
TEST24     00014000   00014000   00000008    Amode 24   Non-reloc
Ready;  T=0.01/0.01 16:32:50
```

Figure 7: LOAD and the PRES option

first program cannot be STARTed, and will not be included in a GENMOD, because it was cleared from the loader tables by the second program. If you are doing this, you probably should be using INCLUDE instead.

Shared segments

Shared segment support is quite different in VM/XA. With large virtual machines, shared segments may have to load within the virtual machine.



Assume that, for example, the VS FORTRAN compiler's shared segment must load at 11M every time you compile a program. If your virtual machine is smaller than 1 1M, the segment loads above the end of your machine, as it does in VM/SP. If your virtual machine is larger than 1 1M, CMS makes room for the segment within your virtual machine.

If the memory needed for that segment is already in use when you try to compile, you will receive an error message. To prevent this error, you may reserve space for the segment with-the SEGMENT RESERVE command. This should be done in the PROFILE EXEC if possible, or the storage may already be in use.

SEGMENT RESERVE has several disadvantages, however. Users must know the name of the segment, which may or may not have a meaningful relationship with the product. The SEGMENT RESERVE command also ties up that storage until explicitly released, preventing any overlapping segment from using that space. That can be a disadvantage if several unrelated products share the same address range, or if test and production versions of a program are defined to overlap (as they usually are).

Some instructions behave differently depending on the AMODE of the executing program. In general, these differences involve addresses as 24-bit or 3 1 -bit quantities.

In AMODE 24, the Load Address (LA) instruction clears the high-order byte to zero. In AMODE 31, it clears only the high-order bit to zero. If you want to clear a known 24-bit -address, use N Rn,=X'00FFFFFF' . This works regardless of the current AMODE.

In AMODE 24, the Branch and Link instructions (BAL and BALR) put the program mask in the high-order byte of the 24-bit return address. In AMODE 31 they put the addressing mode in the

high-order bit of the 3 l-bit return address. If you were using these instructions to obtain the program mask, use the IPM instruction instead. Unfortunately, IPM is an XA-only instruction, so programs that need to do this in any architecture will have to use dual-path code.

A program can switch AMODEs as it executes. It might do this to prepare or process control blocks or parameter lists used by programs in different AMODEs. There are a number of machine instructions that can be used to change AMODE, but the easiest way is to use the AMODESW macro. This macro can also be used to call subroutines or externally-loaded programs, saving, changing, and restoring the AMODE in the process.

```
AMODESW SET,AMODE=24
AMODESW SET,AMODE=3 1
```

The most frequent change needed to an Assembler program is for SSM (Set System Mask) instructions. This instruction is valid in both 370 and XA architectures, but the mask values are different. If an untested program ends with a specification exception, and the location four bytes before the error address contains X'80', then the offending instruction is an SSM (see Figure 8).

```
bomber
DMSITP141T Specification exception occurred at 80014022 in
           routine BOMBER
CMS

#cp display s1401e
R0001401E  80                                        E6
```

Figure 8: Detecting an SSM instruction in XA

The ENABLE macro can be used to enable or disable interrupts in any architecture or release of CMS. You do need the new macro libraries to assemble the macro, but once assembled, the resulting program will run in any release of CMS.

If you had...	Use...
SSM =X'00'	ENABLE INTTYPE=NONE
SSM =X'FF'	ENABLE INTTYPE=ALL

The following 370 instructions will not work in X.4 architecture and will also cause specification exceptions:

SIO, SIOF, TIO, TCH, HIO, STIDC, HDV, CLRIO, CLRCH
ISK, SSK

DIAG 18, DIAG 20

Anyprogram containing these instructions will need more extensive work. Use of the I/O instructions,-in particular, will make a program very architecture dependent. It is worth considering changing the program to avoid these I/O instructions entirely. Use CMS services if possible, or, if you must do channel programming, investigate the new A4 and A8 DIAGNOSE instructions that perform architecture independent channel program I/O.

To use memory above the 16M line, you must make sure that your program does not depend on addresses being 24 bits long . The most common problem is using the previously unused high-order byte-of addresses to store or pass information (see Figure 9). Sometimes getting rid of this problem is a simple matter of giving the address its own fullword, sometimes it requires a complete redesign of the program.

```
         L      R1,ADCON          Get the address
         LA     R1,0(,R1)           and remove the flag.
         ...
ADCON    DS     0A                Address of something.
FLAG     DS     X                 Flag in the high byte.
         DS     AL3               24 bit address.
```

Figure 9: A problem in AMODE 31

To use memory above the 16 megabyte line, you must also use CMS services that work with 31 bit addresses. Many old services do not, and there are new replacements.

If your program used.. .	Now it should use.. .
DMSFREE macro	CMSSTOR OBTAIN macro
DMSFRET macro	CMSSTOR RELEASE macro
LINEDIT macro	APPLMSG macro
RDTERM macro	LINERD macro
WRTERM macro	LINEWRT macro
DMSKEY, DMSEXS macros	IPK, SPK instructions
svc 202	CMSCALL macro (SVC 204)
ASCANN function	SCAN macro
ATTN function	CMSSTACK macro
NUCEXT function	NUCEXT macro
SUBCOM function	SUBCOM macro
WAITRD function	LINERD macro
DIAG 58 (fullscreen)	CONSOLE macro
DIAG 64 (shared segments)	SEGMENT macro
DIAG 20 (I/O)	DIAG A8

Figure 10: New CMS Services

The new services are all at least functional replacements on the old services; many are substantial improvements on the old. However, use of them will prevent your program from running in earlier releases of CMS[2]. As with the architecture dependent machine instructions, you must sometimes make execution-time choices between instructions or macros (see Figure 11).

2. You can obtain an SPE from IBM to add some of these services to CMS 5.0. However, if you distribute programs, you probably can't depend on that SPE being installed at any given installation.

```
          USING  NUCON,RO
          ...
          CLI   CMSPROG,VMSP5    If this is CMS 5.0 (or before)
          BH    CMS55            do
          ...                      CMS 5.0 dependent code.
          B     SKIP1            end
CMS55     DS    OH               else do
          ...                      CMS 5.5 dependent code.
SKIP1     DS    OH               end
          ...
          TM    NUCMFLAG,NUCMXA  If this is 370 architecture
          BO    XAARCH           do
          ...                      370 arch dependent code.
          B     SKIP2            end
XAARCH    DS    OH               else do
          ...                      XA arch dependent code.
SKIP2     DS    OH               end
          ...
          CMSLEVEL
          NUCON
          END
```

Figure I I : *Dual* ***paths for programs***

The conventions for receiving control in a MODULE from CMS have changed somewhat. They are detailed in Figure 12. The major change is that the calltype flag that was 'previously passed in the high-order byte of R1 is now in the save area pointed to by R13 instead. This, of course, is to allow the address in R1 to be a 31 bit address.

RO contains the address of the extended parameter list. It is 24 or 31-bit depending on your AMODE.

R1 contains the address of the tokenized parameter list. It is 24 or 3 1 -bit depending on your AMODE. If it is 31-bit, it does not contain the calltype flag in the high-order byte.

R12 contains the address of the entry point of your program. The high-order bit is 1 if you are in AMODE 31.

R13 contains the address of a save area. This save area now contains the calltype flag. The USERSAVE macro maps this area.

R14 contains the return address. The high-order bit by convention indicates the AMODE of your caller. If called from CMS, this will always be zero, because CMS is basically AMODE 24.

R 15 contains the same information as R 12.

Figure 12: MODULE entry conventions

The conventions for non-module linkage remain basically the same, as detailed in Figure 13, with the addition that AMODE information may be present in the addresses in R14 and R15.

R13 contains the address of a save area.

R14 contains the return address. The high-order bit by convention indicates the AMODE of your caller.

R15 contains the address of the entry point of your program. By convention, the high-order bit is 1 if you are in AMODE 31.

Figure 13: Non-MODULE entry conventions

There is a new CMSRET macro to return from a program to CMS. It allows you to return a return code and other register values, and it takes care of AMODE switching if necessary. However, CMSRET makes your program unable to run in VM/SP. Most of the time, a good, old-fashioned

```
BR    R14
```

does the trick.

Summary

The conversion effort was moderately disruptive to the user community. Four Systems Programmers were involved over the course of a year. A significant number of people from User Services spent time on the conversion, installing new vendor packages, converting local software, writing documentation and assisting users. An unknown amount of effort was also required by individual users. The experimental groups with complex software had a fair bit of trouble in conversion, and the group that required XA exploitation found that a great deal of effort was required (and spent quite a bit of time in my office).

As mentioned previously, a number of local programs were left behind, but often because they were functionally replaced by other programs, or because they were little used. We used the CMAP monitoring program from VM•CMS Unlimited to aid in determining which programs to leave behind. No major vendor packages were left behind, and most would even run in XA architecture virtual machines as well.

In some respects, the conversion was surprisingly easy. Many programs ran with little or no change, even in XA architecture. Most of those that did not convert did not use the standard system interfaces, or depended on system internals. In a number of cases it was simpler to rewrite an ill-behaved program than it was to change it; I took the opportunity to rewrite several assembler programs in C.

SLAC will continue working towards full XA exploitation. The cutover date from HPO to VM/XA is February 21, 1990. After that, more effort will be directed towards changing programs for XA toleration and exploitation.

More work is needed by IBM to move the system towards maturity. VM/SP and VM/XA should be converged towards a common system. VM/XA needs many of the features that VM/SP Release 6 and even Release 5 have. SHARE must continue to work with IBM in defining our requirements for future releases.

BIBLIOGRAPHY

CONVERSION:

GG24-3174 Bimodal CMS for VM/XA Systems
A very high-level summary of the changes in CMS 5.5. It won't tell you anything substantive.

SC23-0357 Conversion Notebook
The title sounds promising, but I wouldn't recommend it for most people. Most of the book is concerned with hardware and network planning and with system generation. However, there is one very useful 10 page section that lists all the differences in CP commands and responses between HPO and XA.

GG24-3278 VM/XA SP Bimodal CMS Application Programming Considerations
This is a well-written and informative manual (it was written by one of the XA system developers). It is mostly for those who write in Assembler, but has some item of interest to Fortran programmers, especially those with complex linking requirements. The topics included are: program handling, program invocation, segment management, storage management, REXX (just a little bit), dual-path code with good examples, migration planning, and loader operation.

SC23-0403 CMS Application Program Conversion Guide
This manual summarizes the changes between VM/SP and VM/XA-SP in both CMS and CP. There is some overlap with the 'Programming Considerations' manual above, but enough different information for it to be worth a read. Again, it is mostly for Assembler programmers. Topics include System/370 compatibility, 370-X-A toleration, 370-XA exploitation, VM/XA SP CMS functional changes, and the VM/XA SP CMS programming interface.

GUIDES:

SC23-0355 CMS Application Program Development Gu'de
This is a detailed guide to developing programs under CMS 5.5. It roughly corresponds to the old (SC24-5286) VM/SP CMS for System Programming manual. It has been totally rewritten and reorganized and has a great deal of useful information about the new architecture. The major topics are using CMS native services, managing CMS programs, and OS/MVS, DOWSE, and VSAM simulation.

SC23-0356 CMS User's Guide
This is the command-level guide to using CMS. It has not changed a great deal from VM/SP release 5.

SC23-0375 System Product Interpreter (REXX) User's Guide
No change from SC24-5238 for VM/SP Release 5.

SC23-0373 System Product Editor (XEDIT) User's Guide
No change from SC24-5220 for VM/SP Release 5.

SC23-0377 Virtual Machine Operation
A guide to running guest operating systems such as MVS or DOS/VSE under VM/XA. Not of interest to most people. It does contain the tutorial on how to use the new TRACE command for debugging, however.

FORTRAN:

SC26-4222 VS Fortran Version 2 Release 3 Programming Guide
Several short sections describe some considerations of running FORTRAN programs in XA, especially in the area of making use of more than 16 megabytes of memory.

REFERENCE:

SC23-0354 CMS Command Reference
The familiar CMS reference manual, now grown to 942 pages! There are a number of new commands, and some changed ones, mostly in the area of program linking, loading, and management.

SC23-0358 CP Command Reference
The CP reference weighs in at 800 pages, and includes all CP commands, both privileged and non-privileged. A number of commands are added and changed, notably: ADSTOP and PER become TRACE, SET and QUERY have changes, and AUTOLOG is incompatible.

SC23-0376 System Messages and Codes Reference
Many new and different messages in the CP section. Many messages now have four-digit message numbers.

SC23-0374 System Product Interpreter (REXX) Reference
No change from SC24-5239 for VM/SP Release 5.

SC23-0372 System Product Editor (XEDIT) Reference
No change from SC24-5221 for VM/SP Release 5.

SC23-0402 CMS Application Program Develonment Reference
This is the replacement for the old (SC24-5284) CMS Macros and Functions Reference, but double the size. There are many new CMS macros to more formally define the programming interface to CMS, and to provide XA capability.

SC23-0370 CP Programming Services
This manual is the reference to all Diagnose instructions and for IUCV. It is the replacement for (SC24-5288) System Facilities for Programming.

SA22-7085 IBM System/370 XA Principles of Operation
This describes the operation of an XA-mode virtual (or real) machine. Some instructions are new and some are removed. All instructions handle 31-bit addresses and many change accordingly. In a 370-mode virtual machine, the previous manual (GA22-7000) IBM System/370 Principles of Operation is still relevant.

SC23-0353 Administration
This replaces (SC19-6224) CP for System Programming. It is only of interest to systems programmers.

MISC:

GC28-1154 MVS/XA Supervisor Services
This manual documents the OS macros used in CMS, such as GETMAIN, SPIE, STAE, etc. CMS 5.5 uses the MVS/XA versions of these macros, while previous versions used the OS/MVT versions. This is now the relevant manual for these macros.

GC26-4014 MVS/XA Data Management Macros
This manual documents the OS data management macros used in CMS, such as DCB, GET, PUT, READ, and WRITE.

GC26-4011 MVS/XA Linkage Editor and Loader User's Guide
This manual documents the usage of the MVS loader, available in CMS as the LKED command.