Inter-Office Memorandum

To	File	Date	October 20, 1978
From	Roy Levin, Chuck Geschke, Jim Mitchell	Location	Palo Alto
Subject	Mesa for Extended Memory Altos (version 3)	Organization	CSL

XEROX

Filed on: [Ivy]<XMesa>Doc>XMesa.bravo

A version of the Mesa system that exploits in certain ways the extended memory features of Alto IIs is now available. This document describes the facilities provided by this system, its limitations, and its compatibility with other current Mesa systems.

The system described in this document is functionally compatible with Mesa 4.1, except as explicitly noted below. However, we have no plans to maintain compatiblity with future releases of the Mesa system.

Overview of XMesa

XMesa uses the extended memory of an Alto II XM as additional swapping space for code. This means that code and data need not co-exist in the primary 64K of memory. XMesa takes advantage of any available extra space automatically; standard Alto programs do not need to be modified to run with XMesa. XMesa does not simplify the task of managing data structures that are too large to fit in 64K, but it does free (most of) the 64K primary address space for use by data.

Because XMesa uses extra memory for code segments, it includes a page-level storage allocator for the additional banks. Client programs may request storage in the additional banks by using this allocator; the interface is public. XMesa provides a primitive mechanism to permit blocks of data to be copied between banks of memory, but gives no other assistance in accessing information in the extended memory banks.

Clients should understand that, while XMesa is interface-compatible with Mesa 4.1, the implementation of certain system components (e.g., the page-level memory allocator and segment swapper) has changed substantially. Client programs that assume undocumented properties of these components do so at their peril. Information about incompatibilities in the implementation appears later in this document.

Compatibility with Mesa 4.1

Considerable efforts have been made to ensure that XMesa is compatible with Mesa 4.1. Exceptions are minor and typically affect only those programs that depend upon the format of global frames (details appear below). To enhance the value of compatibility and because wide-bodied Altos are relatively scarce, XMesa will execute properly (though with some performance degradation) on a normal Alto (I or II) without recompilation or rebinding. Indeed, XMesa requires no special compiler or binder. Both BCDs and image files are compatible across machines. Thus a Mesa disk containing the XMesa versions of `RunMesa.run`, `XMesa.image` (or `BasicXMesa.image` and `XImageMaker.bcd`), and the debugger can be used without alteration on any Alto, XM or otherwise. Specifically,

1) Mesa 4.1 BCDs will load and execute under XMesa,
2) Mesa 4.1 image files will run correctly under XMesa,
3) Image files created with XMesa will run (under XMesa) on any Alto,
4) Files created with Mesa 4.1 and XMesa may be debugged (under XMesa) on any Alto.

Of course, these properties can be guaranteed only for programs that do not assume the existence of extended memory.

Note! Compatibility is assured only for Mesa 4.1. BCDs compiled or bound by previous releases of the Mesa system, including Mesa 4.0, will NOT work properly under XMesa.

The Public Interface to XMesa

To preserve compatibility with Mesa 4.1, XMesa provides its services exclusively through a new interface, XMESADEFS. However, the facilities supplied logically belong in other existing interfaces, notably SEGMENTDEFS and ALTODEFS. Thus, XMESADEFS actually includes several semi-independent interfaces. Unless otherwise stated, everything defined in this section comes from XMESADEFS.

Extended Memory Management

Segments in extended memory are created with the usual primitives in SEGMENTDEFS. However, XMesa recognizes additional "default" parameter values for those procedures that expect a VM base page number. **DefaultBase0**, **DefaultBase1**, **DefaultBase2**, and **DefaultBase3** request allocation in a specific memory bank. **DefaultXMBase** requests allocation anywhere in the extended memory banks (banks 1-3). **DefaultBase** (defined in SEGMENTDEFS) requests allocation anywhere in memory (banks 0-3) *if the segment is a code segment*, otherwise, it is equivalent to **DefaultBase0**.

The following procedures logically belong in SEGMENTDEFS. They convert between segment handles and long pointers, and work for segments anywhere in the 18-bit address space.

```
XVMtoSegment: PROCEDURE [a: LONG POINTER] RETURNS [SegmentHandle];

XSegmentAddress: PROCEDURE [seg: SegmentHandle] RETURNS [LONG POINTER];

XVMtoDataSegment: PROCEDURE [a: LONG POINTER] RETURNS [DataSegmentHandle];

XDataSegmentAddress: PROCEDURE [seg: DataSegmentHandle]
    RETURNS [LONG POINTER];

XVMtoFileSegment: PROCEDURE [a: LONG POINTER] RETURNS [FileSegmentHandle];

XFileSegmentAddress: PROCEDURE [seg: FileSegmentHandle]
    RETURNS [LONG POINTER];
```

The following definitions logically belong in ALTODEFS. They define parameters of the extended memory system.

```
BankIndex: TYPE = [0..3];
PagesPerBank: PageCount = . . . ;
MaxXPage: PageNumber = 1777B;                    -- analogous to MaxVMPage
```

The following signal logically belongs in IMAGEDEFS, and is raised when MakeImage discovers a segment in banks 1-3 that cannot be swapped out. (See the section on restrictions, below, for more information about image files). If this signal is **RESUME**d, MakeImage assumes that the segment in question has been removed from high memory by the client; failure to do so may result in an image file that cannot be restarted.

```
ImmovableSegmentInHighBank: SIGNAL [SegmentDefs.FileSegmentHandle];
```

Long Pointer Support

The following (machine code) procedures return the high and low halves of a long pointer.

```
HighHalfPtr: PROCEDURE [LONG POINTER] RETURNS [CARDINAL] ;

LowHalfPtr: PROCEDURE [LONG POINTER] RETURNS [POINTER] ;
```

XCOPY is analogous to **COPY** in INLINEDEFS, but accepts long pointers instead of pointers and moves data across memory bank boundaries. However, not all possible pairs of pointers are permitted. **XCOPY** raises **InvalidXCOPY** if the long pointers specify transmission between banks **tobank** and **frombank** such that **tobank = frombank** and **tobank**, **frombank = 0**. (In other words, **XCOPY** can only handle one bank other than bank 0 at a time.) **XCOPY** raises **XMNotAvailable** if the pointers are legal but the requested extended memory bank does not exist. (Note: **XCOPY** is a machine code procedure.)

```
XCOPY: PROCEDURE[from, to: LONG POINTER, nwords: CARDINAL] ;

InvalidXCOPY: ERROR;

XMNotAvailable: ERROR;
```

The implementation of XMesa prohibits client programs from manipulating the emulator bank register directly (see section on restrictions, below). Clients wishing to perform a BitBlt across bank boundaries must therefore use the **XBitBlt** procedure, which verifies that the requested extended memory bank is available (raising **XMNotAvailable** if it isn't), then loads the emulator's alternate bank register with **bank** and performs a BitBlt.

```
XBitBlt: PROCEDURE[bbt: BitBltDefs.BBptr, bank: XMesaDefs.BankIndex];
```

The following procedures convert between page numbers and long pointers, and are analogous to **AddressFromPage** and **PageFromAddress** in SEGMENTDEFS. Illegal parameters are detected and cause the indicated signals to be raised.

```
LongAddressFromPage: PROCEDURE[page: AltoDefs.PageNumber]
    RETURNS[lp: LONG POINTER];
```

```
InvalidXMPage: ERROR [page: AltoDefs.PageNumber];

PageFromLongAddress: PROCEDURE[lp: LONG POINTER]
    RETURNS[page: AltoDefs.PageNumber];

InvalidLongPointer: ERROR [lp: LONG POINTER];
```

Configuration Information

XMesa has an internal data structure containing information about the hardware configuration of the Alto on which it is running. Clients may obtain a copy of this data structure by writing

```
memoryConfig: MemoryConfig _ GetMemoryConfig[];
```

and should normally test for the existence of extended memory by examining **memoryConfig.useXM**. The extant banks of memory are indicated by **memoryConfig.banks**, which is a bit mask (*e.g.* **memoryConfig.banks =14B** => banks 0 and 1 exist).

```
MachineType: TYPE = {unknown, AltoI, AltoII, AltoIIXM, D0, Dorado};

MemoryConfig: TYPE = MACHINE DEPENDENT RECORD[
    reserved: [0..37B],
    AltoType: MachineType,
    useXM: BOOLEAN,
    unused: [0..3],
    secondROM: BOOLEAN,
    banks: [0..17B],
    mesaMicrocodeVersion: [0..377B],
    XMMicrocodeVersion: [0..377B]];

GetMemoryConfig: PROCEDURE RETURNS[MemoryConfig];
```

useXM is true if and only if the following conditions hold:

1) the machine is an Alto II with XM modifications (**AltoType = AltoIIXM**),
2) the Alto has more than one memory bank installed (**banks ~= 10B**),
3) the Alto has a second ROM (**secondROM** is true), and
4) the second ROM contains the current version of the XMesa microcode.

The microcode version fields tell only the *microcode* version, *not* the Mesa release number. (For example, for Mesa 4.1, **mesaMicrocodeVersion** is 34.) **XMMicrocodeVersion** will be non-zero if and only if XMesa microcode is installed in the second ROM.

If **AltoType** is **unknown**, all other fields of the record are unreliable. This can happen only if XMesa is initialized by an obsolete version of RunMesa.

How to Obtain and Use XMesa

The files for XMesa reside on [Ivy]<XMesa>. The subdirectory structure parallels that used on [Iris]<Mesa>, but only those files that supersede Mesa 4.1 are stored on <XMesa>. The files on <XMesa> are organized as follows:

<XMesa>
`RunMesa.run`
`XMesa.image`, `.symbols`, and `.signals`
`BasicXMesa.image`,. `symbols`, and `.signals`
`XMesa.cm`, `BasicXMesa.cm`, and `MakeXMDebug.cm`
miscellaneous command files

<XMesa>System>
`XImageMaker.bcd` and `.symbols`
system definitions and program modules (`.mesa` and `.bcd`)
source and object code for RunMesa and microcode (mostly in `.dm` files)
miscellaneous command files

<XMesa>Doc>
documentation for XMesa

<XMesa>XDebug>
modules to be installed in the Mesa 4.1 debugger (`.mesa` and `.bcd`)

`RunMesa.run` is required by all XMesa programs and will also run Mesa 4.1 programs without alteration. *Failure to use the XMesa version of RunMesa will cause XMesa programs to behave in unpredictable ways.*

The microcode that must be installed in the second control ROM is stored in `<XMesa>System>XMesaROM.mb`. This file is in suitable form for use by the PROM blaster.

`XMesa.cm` and `BasicXMesa.cm` retrieve the corresponding image files, the debugger, and related supporting files. `MakeXMDebug.cm` installs the debugger, including the modules required for debugging the XMesa environment. (`XMesa.cm` and `BasicXMesa.cm` implicitly execute `MakeXMDebug.cm` as well.)

Restrictions, Limitations, and "Features"

Images and Checkpoints. MakeImage cannot preserve the contents of extended memory in the image file it constructs. If MakeImage is invoked on an Alto II XM, it will swap out all unlocked file segements in extended memory. (It will also move any locked code segments to primary memory.) If any segments then remain in extended memory, MakeImage will refuse to build the image file. Analogous comments apply to CheckPoint.

Bank Registers. XMesa assumes it has exclusive control of the emulator bank register. Client programs must not attempt to alter the bank register, but rather must use the public interfaces for moving data to and from extended memory (see **XCOPY** and **XBitBlt**, above).

Global Frame Format. The format of global frames and segment objects has changed slightly, although (for compatibility) CONTROLDEFS and SEGMENTDEFS have not been recompiled. The formats assumed by XMesa appear in GLOBALFRAMEDEFS and XMESAPRIVATEDEFS, respectively. The only change likely to affect client programs is the removal of the code segment handle from the

third word of a global frame. Programs that previously assumed this pointer's presence in the frame should now import CODEDEFS and replace

frame.codesegment

by

CodeDefs.CodeHandle[frame] .

Segment Alignment. Segments may not cross bank boundaries.

Resident Code. The amount of resident code in XMesa is 6 pages greater than in Mesa 4.1. Accordingly, XMesa may exhibit poorer swapping performance on non-XM Altos than Mesa 4.1 does.

Old Mesa System Modules. Any Mesa BCD for which an XMesa version exists must not be used under XMesa. No check for inappropriate BCDs is made and unpredictable results may occur. BasicXMesa users: if you make image files, be certain to obtain `XImageMaker.bcd`.

Debugger Incompatibilities. The Mesa 4.1 version of XDebug cannot be used with XMesa unless the module `XMDebug.bcd` is loaded when the debugger is installed. In addition, the debugger's CoreMap command does not work at all under XMesa. The UserProc `XCoreMap` should be installed in the debugger and used instead. The command file `MakeXMDebug.cm` will install the debugger properly for use with XMesa.

Microcode ROMs. RunMesa determines whether the microcode contained in the Alto's second control ROM (if any) is suitable to run XMesa, and will set **memoryConfig.useXM** accordingly. (See the section entitled *Configuration Information*, above.) However, old versions of RunMesa (before 29.16.3) do not contain the necessary safety checks, and will cause XMesa to execute incorrectly on a machine that contains improper microcode (**useXM** will erroneously be **TRUE**). This will cause unpredictable results. XMesa clients should ensure that they have an up-to-date version of `RunMesa.run`. In particular, if XMesa is loaded by RunMesa 29.16.3, it can execute successfully on an Alto II XM with standard Mesa 4.1 microcode in the second ROM, but it cannot take advantage of the additional memory banks. If XMesa is loaded by an older version of RunMesa on such a machine, it will behave unpredictably.

Microcode in the RAM. The microcode to support XMesa occupies slightly more than 1K of control memory space. Most of it resides in the second 1K of control ROM (ROM1), with the excess overflowing into the RAM. The remaining RAM space is available to users who wish to write their own microcode, but certain conventions must be observed. Those who need to execute special microcode from Mesa programs should consult [Ivy]<XMesa>Doc>Microcode.press (and .bravo).

Swapper Algorithms. The XMesa swapper loads a segment into extended memory by first swapping it into primary memory, then copying it to extended memory and releasing the primary memory space. Thus, if primary memory is so full that the requested segment cannot be swapped in, **InsufficientVM** will be raised, even though sufficient space for the segment may exist in other banks. (Analogous comments apply when swapping out segments that must be written to disk.)

Distribution:
CSL
Mesa Group