OS-9 GMX™ III Manual Addenda
Version 1.2, 09/14/83

March 84

This document describes features specific to OS-9 GMX™ III, Version 1.2 (the GIMIX version of OS-9 Level II, for the GMX™ 6809 CPU III board), that are not documented in the standard OS-9 USERS and/or SYSTEM PROGRAMMER'S manuals. It also describes some important differences between this version of OS-9 and the earlier versions.

CONTENTS

* Denotes IMPORTANT variations from versions 0.0 and 0.5 or from the information in the standard OS-9 manuals. Please read before using this version.

Rev.F

***** Differences Between Versions 1.1 and 1.2 *****

Except for the changes listed in this section, OS-9 GMX III, V1.2 is fully compatible with Version 1.1.

KERNEL: Memory modules may now be loaded into memory space that is not contiguous. This allows better utilization of available memory.

A "Suspend State" has been added to the possible process states. This allows (but does NOT require) drivers to "Suspend" the current process while awaiting an interrupt. In earlier versions, the process was "put to sleep" and a signal sent to "awaken" it. The Suspend State decreases the amount of overhead during interrupt processing. NOTE: Older drivers, using the "sleep" method will still work properly with Version 1.2; however, drivers using the "suspend" method will not work with the older versions.

RBF: A byte-locking style of record locking is now used in place of the sector-locking method used with Version 1.1. See Appendix b for more information on record locking. Several other minor bugs and speed enhancements have also been made to RBF.

DEVICE DRIVERS: The device drivers (with the exception of G68 floppy disk driver) have been modified to take advantage of the "Suspend State" described above. G68 does not use the "Suspend State" in order to retain the "drive motor time-out" protection described in the G68 Driver section. G68 has been modified to eliminate the "Error #210" problem encountered on earlier versions.

UTILS: In addition to minor changes to some of the utilities, MDIR and PROCS have been modified to work with non-contiguous memory modules. The new versions of these programs MUST be used with Version 1.2. The new versions will NOT work properly with the old Kernel.

PROGRAM DEBUGGING: When a UAM or WPT trap (error #199 or #198) is caused by a program running under the OS-9 debugger, control will now return to the debugger to facilitate location of the problem that caused the trap. A WDC trap (error #197) will still cause the system to return to "shell".

***** RBF Changes *****

Due to changes in RBF affecting the bitmap and directory handling, disks created under GMX™ III, V1.1 or V1.2 (or Level I, V1.2/Level II V1.1/2) CANNOT be written on by the versions of RBF used in Versions 0.x of OS-9 Levels I or II. V1.2 disks can always be READ by these older versions of RBF. V1.2 can read and write disks created by old RBFs; however, once written on by V1.2 RBF, the disks should not be written on by the old RBF. Note: This change affects all current implementations of OS-9 Levels I and II and is not specfic to the GMX™ III version.

OS9defs Files

The "OS9defs" files have been revised. They have been split into several different files to allow selective inclusion in assembler programs, depending on the intended application. A special file (li.equates) may be included in existing programs, that use the old short-form names, to equate the old names to the new, longer names used in the new "OS9defs" files.

***** Interrupts *****

Since OS-9 is an interrupt driven system, all I/O devices (including the time-of-day clock, disk controller(s), and serial/parallel I/O boards for terminals and printers) must have their IRQ interrupt outputs enabled.

Most devices do not actually generate an interrupt on the bus until initialized by OS-9; and a system reset (or turning off the power) clears any interrupt output that may be pending: once the device has been initialized. However, the 6850 ACIAs (used on GIMIX 1-port, 2-port and 8-port serial interfaces) have no direct connection to the system reset line. They can only be reset by the software, or by turning the power off. Because of this, it is possible to reset a running system and leave unserviced interrupts (from the 6850s) on the bus. These "left-over" interrupts can interfere with proper operation of the system when it is rebooted after a reset. Since OS-9 does not automatically initialize all of the serial ports when it is booted (normally only "TERM" is initialized), the "left-over" interrupts, if any, must be cleared by some other means. Turning the system off before booting will insure that the interrupts are clear, but this is not always practical. The INIZ command should be included in the startup file to clear any 6850 interrupts that may be left after a reset.

INIZ Command

The "INIZ" command is used to clear any pending interrupts from 6850 ACIAs when the system is rebooted after a reset. If these interrupts are present and not cleared when the system is booted, a variety of problems, including a marked reduction in system speed (throughput) or the inability to boot the system at all, may result.

SYNTAX: INIZ (devname) [(devname) (devname) ...]

INIZ clears any pending interrupts on the device(s) listed by opening and then closing a path to each of them. The list of devices should include ALL of the 6850 type serial devices (normally T1, T2, ... Tn, and P1) in the system, with the exception of "TERM" which is automatically reinitialized by the system. Only those devices that are included when the system is booted (devices in the OS9boot file) should be listed. If no devices are listed, INIZ attempts to read a device list from the STD input path.

EXAMPLE: INIZ /T1 /T2 /T3 /P1

If INIZ cannot open one of the devices, an error is returned and iniz terminates without further processing of the device list.

COPY Command

In addition to the changes noted in the "User's Manual", the copy command has been modified to preserve the "creation" and "last-modified" dates of the copied file. The new "COPY" will also preserve the file's owner ID if you are user number zero. To work properly, this version of the copy command (edition 7 or later) must be used with edition 11 or later of RBF.

OS-9 GMX™ III and the GMX™ CPU III allow a full 64K of RAM to be installed on each of the first 15 memory banks ($0-$E). The 16th bank ($F) can have up to 56K of RAM installed; the upper 8K are reserved for I/O and the operating system PROMs. RAM should be installed in contiguous 64K banks beginning at address $0000 on bank $0. All I/O devices including the motherboard and disk controllers must have extended address decoding enabled and set to decode bank address $F (see appendix a).

Since the GMX™ CPU III automatically switches to the system state when servicing interrupts, the interrupt vectors need not appear in each of the 16 memory banks as in previous systems. This means that up to 64K of RAM may be allocated to a user or task with no system overhead; the ENTIRE 64K is available to the user/task.

Dynamic Address Translation and Memory Attributes

The GMX™ CPU III includes an expanded Dynamic Address Translator (DAT) which, in addition to providing translation in 2K segments for more efficient memory usage, allows the assignment of "memory attributes" on a block by block (2K block) basis. These memory attributes are: Write Protect, Single Step, and Unallocated Memory. They may be selected separately or in combinations for each individual 2K memory segment in the address space. The purpose of these attributes is to provide protection for the system and to provide enhanced debugging capabilities. Currently only the Write Protect and Unallocated Memory attributes are implemented in OS-9 GMX™ III.

Write Protect Attribute

The WPT attribute is used to protect areas of memory, such as those containing sharable OS-9 modules or user programs, from unauthorized and/or unintentional modification. This greatly increases the security of the system by preventing, for example, one user from corrupting the copy of BASIC09 that is being shared by with users. OS-9 determines the intended status of a program (protected or unprotected) by testing a bit in the module header when the module is loaded into RAM. Once loaded, a protected module cannot be written to or modified, except by replacing it with another module.

Normally, all of the modules and programs supplied by GIMIX have their write-protect status set to "protected". Since the system prevents even intentional modification of protected modules, it is not possible to use the Debugger to make changes to a write-protected module. In order to permit modifications to certain modules, such as device descriptors, source files for these modules are included on the system disk(s). Any necessary modifications must be made to the source files, which can then be assembled and installed.

To facilitate debugging, modules can be assembled and run without write protection. However, to maintain maximum system security, they should be write protected once the debugging is completed. Note: It is

advisable, when debugging a program, to test it first with the write protection enabled, to protect the system from crashes (especially if there are other users) and to help detect certain types of bugs.

The write protect status of a module is determined by bit 6 (previously undefined) in the module header's attributes/revision byte (module offset $7). (Bit 7 of this byte is the reentrant/sharable bit, see the System Programmer's Manual.) If this bit is cleared (0) the module will be write-protected when loaded by OS-9, if set (1) the module will be unprotected.

If the GMX™ CPU III detects an attempt to write to write-protected location in memory, a trap occurs and, as a result of the trap, OS-9 will close down the offending task (the task that initiated the write operation) and issue an error message (Error #198).

Unallocated Memory Attribute

The UAM attribute is used to prevent a user/task from making memory accesses (read or write) outside the memory area assigned to it by the operating system. Like the WP attribute, this increases the security of the system by preventing unauthorized access to selected areas of memory. The UAM attribute is automatically controlled by OS-9. When a task is initiated, all memory requested for the task is "allocated" to that task, all memory outside the requested area is "unallocated" and therefore inaccessible to that task.

If the GMX™ CPU III detects an attempt to access (read or write) unallocated memory, a trap occurs and, as a result of the trap, OS-9 will close down the offending task (the task that initiated the access) and issue an error message (Error #199).

Watchdog Counter

An additional function on the GMX™ CPU III board, the Watchdog Counter, is also implemented in OS-9 GMX™ III. The WDC is controlled by the operating system and, when enabled, limits the length of time that interrupts may remain masked and therefore unserviced. The WDC is enabled by OS-9 when the system is switched from the system state (operating system) to the user state (user program running). It begins counting CPU clock cycles when an interrupt occurs and, if the interrupt is not serviced by the CPU in a specified number of cycles (hardware selected: 128 standard,32 optional) a trap occurs. When a WDC trap occurs, OS-9 will close down the offending task and issue an error message (Error #197).

The WDC trap provides two types of protection. It protects against a user program that masks IRQ interrupts; necessary to the system for task switching and I/O processing. It also protects against the execution of certain illegal opcodes which can lock the 6809 in a state in which it will not execute further instructions or respond to any interrupts. To recover from this processor state, the WDC generates a special reset (only the 6809 is reset). The reset is processed through a special trap-vector instead of the normal reset vector, allowing OS-9 to resume processing of all but the task which caused the trap.

Three utility programs are provided with the OS-9 GMX™ III system to demonstrate and test the function of the WPT, UAM, and WDC traps. The utilities (test199uam, test198wpt, and test197wdc) are provided in both source and object form to help the user understand the function of the traps. Execution of one of the utilities will cause OS-9 to close down the task and issue the appropriate error message.

TEST199UAM

Test199uam tests the function of the Unallocated Memory trap by reading from a memory location beyond the end of its normally allocated data area. Note: if test199uam is run with a memory-size modifier in the command line (i.e. test199uam #6K) a trap will not occur and the program will loop until "KILL"ed.

TEST198WPT

Test198wpt tests the function of the Write Protect trap by writing to itself. Since bit 6 of the program's attributes field is clear (0), the module is write-protected and a trap will occur.

TEST197WDC

Test197wdc tests the function of the Watchdog Counter by setting the IRQ mask and entering a wait loop. Once the watchdog count expires (normally 128 CPU cycles) the trap will occur. Note: in order for this test to function, an IRQ must occur after the program is executed; to start the watchdog count. If the system clock has been started (by running "setime") the clock interrupts will cause the trap to occur. If the system clock is not generating interrupts, an interrupt can be caused by hitting a key on the terminal.

In addition to providing a considerable degree of protection for the system, the attributes and watchdog traps also assist in software debugging. In many cases they will detect common problems (caused by uninitialized pointers, incorrect addressing modes, etc.) that might otherwise go unnoticed or be difficult to identify.

Device Descriptors

The device descriptors provided with OS-9 GMX™ III follow the conventions established in earlier versions of OS-9 from GIMIX. The header on the catalog printout, included with the system disk, lists the configuration of the disk, terminal, and printer descriptors provided. Since the descriptors are write-protected by the GMX™ III system when they are in memory, the only way to modify them for user-specific requirements is to modify the source code and reassemble them. The GMX™ III system does not permit "hot patching" with debug as in previous versions. The source code to all of the standard device descriptors is provided in the "SOURCE" directory on the system disk(s).

***** Device Descriptor Changes *****

Due to changes in the Device Descriptors (Device Controller Address field), device descriptors from earlier versions of OS-9 (Level II V0.5 or earlier and OS-9 level I prior to version 1.2) CAN NOT BE USED, without modification, with OS-9 GMX™ III. In order to use existing descriptors, the extended address byte (at module offset $E) must be set to $0F. Without this modification, the system will appear to function, but performance will be seriously degraded!

***** X-ON/X-OFF *****

In addition to the address change noted above, the descriptors for ACIA devices (TERM, T1, T2, ... , and P1) have additional bytes added to control the X-ON/X-OFF functions (module offset $2A = X-ON, offset $2B = X-OFF). These bytes can be set to the desired ASCII codes for these functions if they are required. Setting these bytes to $00 disables X-ON/X-OFF (terminal descriptors shipped with GMX™ III have these bytes set to $00). X-ON/X-OFF can also be enabled using TMODE. When X-ON is set to $11 (control-Q), the QUIT character, normally control-Q, (module offset $23) must be set to some other control character. The recommended substitute character is $05 (control-E). Note: When terminals that generate X-ON/X-OFF sequences are used, X-ON/X-OFF must be enabled in the system.

G68 DMA Disk Controller Drivers

OS-9 GMX™ III uses an enhanced version of the G68 device driver. This G68 supports 3 ms. stepping rates for 5.25" disk drives, allowing faster access times with drives that are capable of stepping at this rate. (All 5.25", 80 track (96TPI) drives currently supplied by GIMIX are capable of stepping at 3 ms.) This option is controlled by a separate bit in the floppy disk device descriptors as described below.

The new G68 prevents the drive motors from timing-out if a drive with no disk (and no "Drive Ready" line) is accessed. On earlier versions of G68, if the drive motors timed-out the system would hang and usually require re-booting. This feature will only function if the system interrupt clock has been started by using the "SETIME" command. We recommend that "SETIME" be included in the "STARTUP" file ("setime 00" will start the clock running, without prompting for new values). Note: To prevent the system from hanging if a non-existant drive is accidentally accessed, a system disk should be created that does not have device descriptors for the non-existant drive(s).

G68 Device Descriptors (D0,D1, etc.)

The device descriptors for the G68 driver follow the definitions given in the OS-9 manuals with the following exception.

MODULE OFFSET	NAME	DESCRIPTION
$14	IT.STP	Bits 0 (LSB) and 1 determine the basic stepping rate as shown in the 179x table in the manual. Bit 7 (MSB) controls "fast stepping" for 5.25" floppy drives

The fast stepping bit (Bit 7) when set (1) in a descriptor for 5.25" drives causes the stepping rate (determined by bits 0 and 1) to be doubled. The 5.25" drive will be stepped at the rate shown in the 179X, 8" drive column in the table. When Bit 7 is clear (0), the drives will be stepped at the normal 5" rate shown in the 179X, 5" column. The fast stepping bit has no effect on 8" drives and should be clear (0) in 8" descriptors.

Caution: Be sure the drives are capable of stepping at the selected stepping rate. In general the only 5.25" drives capable of stepping at 3 ms. (IT.STP = $83) are the newer 80 track (96TPI) drives. Various combinations of the step rate and fast stepping bits can be used to match the stepping rate to the drive(s) used. See the disk drive manufacturer's literature to determine the stepping capabilities of the drive(s). Attempting to step a drive faster than its specified maximum rate will cause excessive disk errors and/or prevent the drive from working at all.

Hard Disk Driver and Descriptors

Like floppy disk drivers, the hard disk drivers consist of two parts; the device driver (XBC) that interfaces the hardware to the Random File manager (RBFman) and the device descriptors (H0 and H1) that describe the characteristics of the hard disk drives. The driver and descriptors are factory configured for the users drive(s) and are not normally modified by the user. NOTE: Version 1.2 of OS-9 GMX™III uses a different driver/descriptor format than V1.1. Drivers/Descriptors from V1.1 should not be mixed with those of V1.2.

Formatting Hard Disks

The same FORMAT program is used to FORMAT both floppy and hard disks. There are several differences in the options available when formatting hard disks, as described below. Hard disks are normally formatted only once; when the drive is first used. Once the hard disk is formatted it does not require reformatting unless the file structure is damaged by a hardware or software fault, or if it becomes desirable to erase ALL of the files on the disk at once.

CAUTION !!

A physical format (see below) destroys all data recorded previously on the disk. A logical format, while it does not completely destroy previously recorded data, makes the data very difficult if not impossible to recover!

Hard disk systems are normally shipped with the disk(s) already formatted by GIMIX. This formatting is done as part of the standard testing performed on the system. The disk may also have copies of the files provided on the system (floppy) disk. Before formatting the hard disk(s) use the directory (DIR) command to check the contents of the hard disk(s). If the disk(s) are already formatted they can be used as-is or reformatted as desired. If reformatting is desired, using the logical-only format option (see below) will save time and accomplish the same results as a complete physical and logical reformatting. Note: the logical-only format option can be used to reformat disks that were previously formatted for a different operating system (such as FLEX).

When FORMAT is called with /H0 or /H1 as the device to be formatted, it first prints a list of the parameters that will be used to determine the format of the hard disk. Unlike the floppy disk format, these parameters are fixed by the requirements of the drives and cannot be changed from the FORMAT utility. Entering either "N" (NO) or "Q" (QUIT) at this time will abort the program and return to the calling program (normally shell). If "Y" is entered, FORMAT will ask :

Both Physical and Logical Format ?

Answering "Y" (YES) will cause FORMAT to perform both types of format.

If "N" (NO) is entered only the logical format will be performed.

PHYSICAL FORMAT

The physical format is normally only required when formatting a new hard disk that has not been formatted previously. The physical format records the information required by the controller to divide the disk into sectors and to locate a particular track and sector on the disk. The operating system does not modify this information once it is written.

LOGICAL FORMAT

The logical format records the information that OS-9 requires to store and keep track of the data files that will be written to and read from the disk. Some of this information is modified by the operating system as files are written to and deleted from the disk. Performing a logical format has the same effect as deleting ALL files on the disk. The logical format takes considerably less time than the physical format.

After the type of format desired is entered, FORMAT will prompt for a disk name, which must be entered following the same syntax as the name for a floppy disk. The next prompt is:

```
Physical Verify Desired ?
```

Answering "Y" will cause a physical verify to be performed. Each sector is read and the information written there is verified. As the verify is performed, FORMAT prints the number of each completed track on the standard output device. Note: Before performing a physical verify, use TMODE to turn the pause function off or FORMAT will stop at the end of each page of track numbers. Answering "N" causes the physical verify to be skipped.

Booting From The Hard Disk

It is possible to boot OS-9 GMX III from a hard disk in one of two ways, depending the type of OS-9 used. If the Support ROM version of OS-9 (GMX IIIs) is used, it can be booted directly from the hard disk, once an "OS9Boot" file and the other necessary files have been installed on the hard disk. (See the Support ROM documentation for more information.) If the original OS-9 (GMX III/without Suppport ROM) is used, the OS9Boot file must reside on a floppy disk. However, OS9Boot can automatically transfer to the hard disk once it has loaded from the floppy. (This method can also be used with the Support ROM version.) The initial directories will be /H0 and /H0/CMDS, the system will attempt to execute the "STARTUP" file from /H0, and "Login" will expect to find the "PASSWORD" and "MOTD" files in the directory /H0/SYS.

In either case, a special OS9Boot file is required. The requirements for the file are the same, regardless of the OS-9 used. To facilitate creation of this file, the files OS9Boot.core and Init.hd are included on the system disk. OS9Boot.core consists of all of the modules normally included in OS9Boot except for the device descriptors (The "Pipe" descriptor is included), the hard disk driver (XBC), and the init module. (Use the Ident command, -s option, to list the modules contained in OS9Boot.core.) Init.hd is a special version of Init for the hard disk.

Before attempting to create a system that will boot from a hard disk, use one of the procedures outlined in the OS-9 manuals to create backups of the original system disk and store the original in a safe place.

To create the special OS9boot, you must use the "OS9gen" command to combine OS9Boot.core, Init.hd, XBC, and at least three device descriptors, D0, H0, and TERM. Additional device descriptors and modules can be included as required, but these six MUST be included or the disk will not boot. Note: For the Support ROM version, OS9Boot.core MUST be the first file included in OS9Boot. The "SAVE" command must first be used to create files containing copies of D0, H0, TERM, XBC, and any additional device descriptors that are required. Note: if changes to the standard device descriptors (such as changing drive stepping speeds or enabling X-ON/X-OFF for terminals) or additional descriptors are required, the appropriate source files can be modified and assembled to generate the necessary descriptors.

The following example outlines the procedure for a system that includes one floppy disk (D0), one hard disk (H0), two terminals (TERM and T1), and a parallel printer (P). The files OS9Boot.core and Init.hd should exist in the current working directory. Note: If the destination for the OS9Boot file is not a freshly formatted disk, and already contains files, OS9Gen may fail because the OS9Boot file must reside on contiguous sectors. When creating a bootable floppy and only one floppy drive is available, it is recommended that the necessary files be copied

to the hard disk so that an empty floppy can be used. When creating a bootable hard disk, the work files should be on the floppy and the hard disk should be empty. An OS9Boot can be created on a disk (floppy or hard) that already contains files; however, it may be necessary to "use up" existing small groups of sectors, by creating dummy files, until a large enough block of contiguous sectors is available. Once a successful OS9gen has been run, the dummy files can be deleted to restore the disk space.

```
OS9:SAVE  D0          Create a file containing the descriptor "D0"

OS9:SAVE  H0          Create a file containing the descriptor "H0"

OS9:SAVE  TERM        Create a file containing the descriptor "TERM"

OS9:SAVE  XBC         Create a file containing the driver     "XBC"

OS9:SAVE  T1          Create a file containing the descriptor "T1"

OS9:SAVE  P           Create a file containing the descriptor "P"

OS9:BUILD HARDBOOT    Create a text file containing a list of the
                      files to be included in the new OS9Boot file.

    ? OS9Boot.core    Must be the first file
    ? Init.hd
    ? TERM            Additional descriptors or modules can be
    ? XBC             included in the list as necessary.
    ? H0
    ? D0
    ? T1
    ? P
    ? [return]

OS9:OS9gen /H0  <HARDBOOT   generates OS9Boot on the hard disk
                             (for Support ROM systems only!)
          - or -

OS9:OS9gen /D0  <HARDBOOT   generates OS9Boot on the floppy
                             (for original or Support ROM
                              systems.)
OS9:
```

The disk created by this procedure can now be used to boot the system. The root directory of the hard disk (H0) must contain a CMDS directory, which becomes the execution directory, and the "STARTUP" file if one is to be used. The system will also search "H0" for the "SYS" directory containing the "PASSWORD" and "MOTD" files.

OS-9 ERROR CODES

The following error codes should be added to the error code list in the OS-9 manuals. The first three are generated by hardware "traps" on the GMX™ III 6809 CPU board, the remaining three are associated with record locking. See Appendix b for more information on record locking errors.

HEX	DEC	
$C5	197	WATCHDOG COUNTER TRAP - The CPU was unable to respond to an interrupt before the Watchdog count expired. The interrupts were masked for too long in a user task or the CPU executed an illegal instruction and was unable to respond.
$C6	198	WRITE PROTECT TRAP - An attempt has been made to write to a module that has its write-protect attribute enabled.
$C7	199	UNALLOCATED MEMORY TRAP - An attempt was made to access (read or write) memory not allocated to the program.
$FC	252	SEGMENT LOCKED - The desired segment is locked by another user.
$FD	253	FILE BUSY - Requested file is non-sharable and busy.
$FE	254	DEADLOCK - Request would cause a deadlock if permitted.

SWITCH CONFIGURATION DRAWINGS FOR OS-9 GMX II & III

The following drawings show the standard DIP-switch configurations for the GIMIX 64K RAM board(s), #68 DMA Disk Controller, Hard Disk Interface (SASI), and Mother Board; when used with OS-9 GMX II & III.

The disk controller(s) must be set to both drive and decode extended addressing (both "ENA" switches ON). The motherboard must be set to decode extended addressing. The boards are addressed so they appear only on bank $F (A16, 17, 18, 19 = ON), at the appropriate base address.

The memory boards are addressed for either 64K banks (OS-9 GMX III systems or OS-9 GMX II systems with modified #05 CPU boards) or 56K banks (unmodified OS-9 GMX II systems). The configuration of switch S3 will normally be the same on all boards in the system, with section-7 ON for 64K banks or OFF for 56K. Both extended address enable switches ("XON"-sections 1 and 6) must be "ON" on all boards. The remaining eight sections of S2 determine the bank address of the board. The boards are divided into two halves, with sections 2,3,4, and 5 used to set the bank address for one half and sections 7,8,9, and 10 the other. In this application both halves are set to the same bank address. The switches are set in a binary pattern with section-2(7) being the least significant bit and section-5(10) the most significant. See the drawing for examples. The boards should be addressed on consecutive banks with the first board on bank $0, the second on bank $1, etc.

***** CAUTION *****

In addition to the boards shown in the drawings, any other memory-mapped boards installed on the 50 pin bus (GIMIX 8-port Serial Interfaces, Intelligent Parallel Interfaces, PROM/ROM boards, etc.) must be capable of extended addressing. Normally, I/O-type boards must be addressed on bank $F, with a base address in the $E000-$EFFF range. Memory-type boards (PROM/ROM) can be addressed as appropriate to the application, as long as extended addressing is enabled. Note: Standard versions of OS-9 only search the lower 56K of bank $F for PROM/ROM memory modules. In order to be located by OS-9, boards containing OS-9 modules in PROM/ROM must be addressed on bank $F.

***** NOTE *****

The switch configuration shown for the #68 DMA board assumes that the system will be booted from a 5.25" drive (D0). If D0 is an 8" drive, S2 section-9 must be OFF.

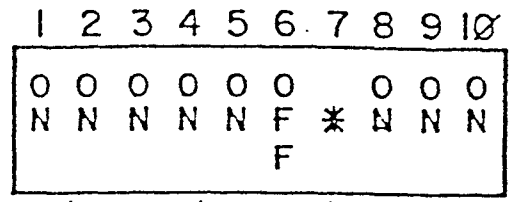
64K RAM BOARD
SWITCH CONFIGURATION
FOR OS-9 GMX II & OS-9 GMX III

BANK ADDRESS (S2)

	$Ø	$1	$2	$3		$F	
1	ON ←				→	ON	XON
2	OFF	ON	OFF	ON		ON	A16
3	OFF	OFF	ON	ON		ON	A17
4	OFF	OFF	OFF	OFF		ON	A18
5	OFF	OFF	OFF	OFF		ON	A19
6	ON ←				→	ON	XON
7	OFF	ON	OFF	ON		ON	A16
8	OFF	OFF	ON	ON		ON	A17
9	OFF	OFF	OFF	OFF		ON	A18
1Ø	OFF	OFF	OFF	OFF		ON	A19

BOARDS SHOULD BE ADDRESSED ON SUCCESSIVE BANKS - (1ST BOARD, BANK Ø; 2ND BOARD, BANK 1; 3RD BOARD, BANK 2; ETC.)

S3



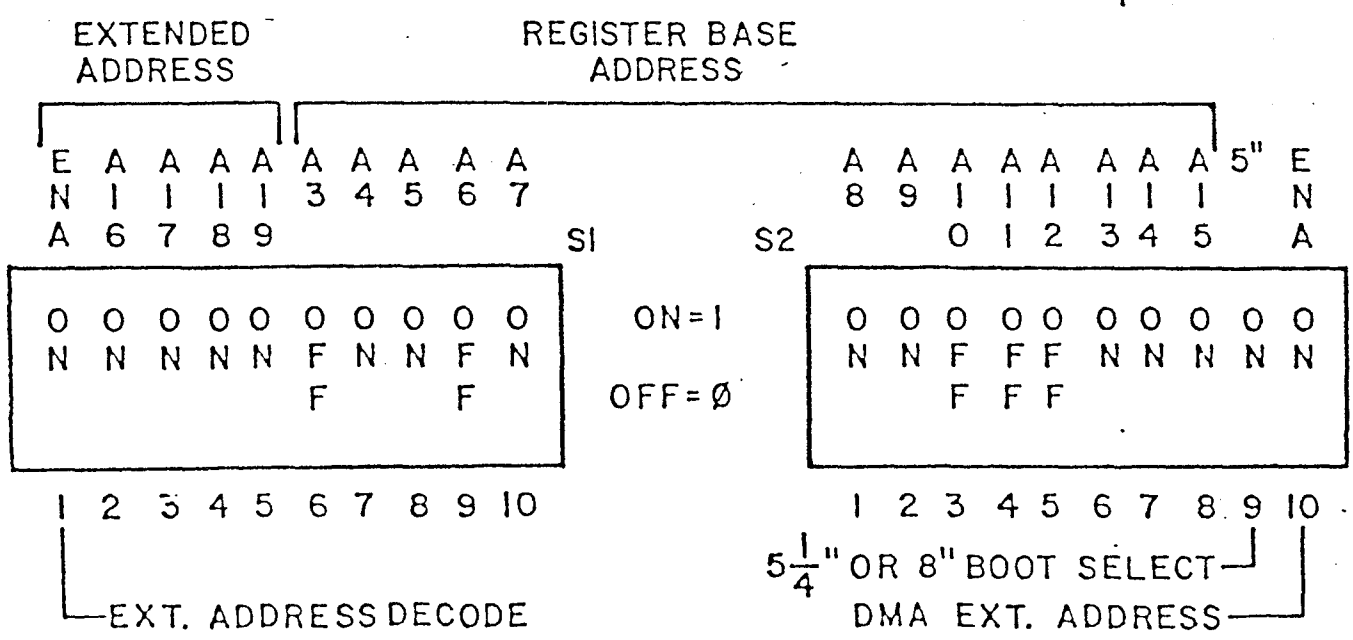
$ØØØØ-$DFFF ($FFFF)

USE 64K FOR ALL BOARDS (EXCEPT BANK $F) IN GMX III AND MODIFIED GMX II SYSTEMS.

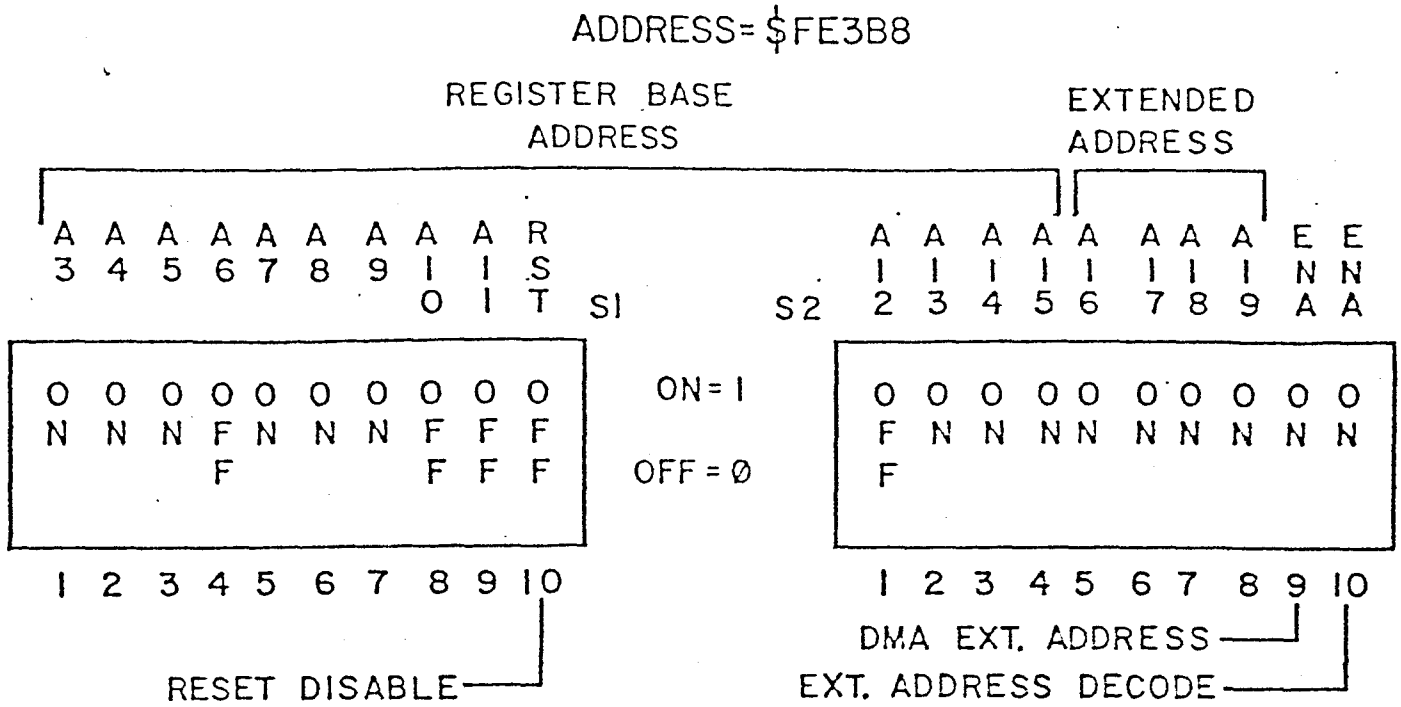
USE 56K FOR UNMODIFIED GMX II SYSTEMS.

DMA FLOPPY DISK CONTROLLER
OS-9 GMX II & GMX III CONFIGURATION

ADDRESS = $FE3BØ BOOT DRIVE (0) = 5 1/4"

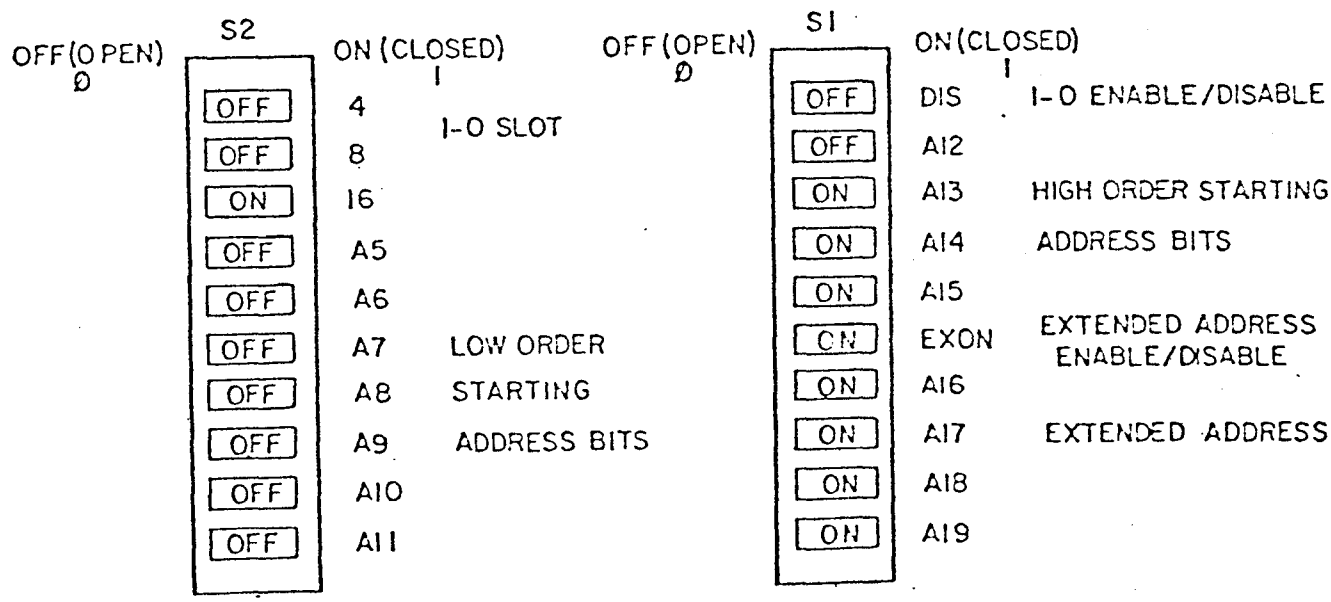


HARD DISK INTERFACE
SWITCH CONFIGURATION
OS-9 GMX II & GMX III



MOTHER BOARD SWITCH CONFIGURATION
OS-9 GMX II & GMX III

ADDRESS = $FE000



APPLICATION NOTE 2
RBF Record Locking Application Notes

Record Locking is a general term that refers to mechanisms designed to preserve the integrity of files that may be accessed by more than one user at a time. The OS-9 implementation of record locking was written to be as invisible as possible to application programs. This has two advantages: existing programs do not have to be rewritten to take advantage of record locking facilities, and most new programs may be written without special concern for multi-user activity.

Simply stated, record locking involves recognizing when a user wants to read a disk record that may be in the process of being modified by another user, and then deferring the read until the record is "safe". This is referred to as conflict detection and prevention. RBF record locking also takes care of non-sharable file locking and deadlock detection.

Record Locking and Unlocking

Conflict detection must notice when a record may be updated. RBF edition 16 and above provides true record locking on a byte basis; earlier versions locked out all sectors in the locked out area. A typical record update sequence is:

```
OS9 I$Read    program reads record        RECORD IS LOCKED
   .
   .          program updates record
   .
OS9 I$Seek    reposition to record
OS9 I$Write   record is re-written        RECORD IS RELEASED
```

If a file is open in update mode, RBF can not determine in advance whether a program is going to update a particular record that has been read. Therefore, ANY read (in update mode) will cause the sector(s) read to be locked out. The record will stay locked until the next Read, Write, or Close occurs. This is how automatic record locking occurs. Files opened in read or execute mode are unable to update records, and therefore do not cause any of the file to be locked out.

A subtle but nasty problem exists for programs that interrogate a data base, occasionally updating its data. If a user looks up a particular record, that record could remain locked out indefinitely, or until the user releases it. This problem is characteristic of record locking systems, and can be avoided by careful programming.

It should be noted that only one portion of the file may be locked out at a time. If a particular application requires more than one segment to be locked out, multiple paths to the same file may be opened, each having its own segment locked out. If this is done, RBF will notice that the same process owns both

paths, and will keep them from locking each other out.

Non-Sharable Files

File Locking may be considered a special case of record locking, in which the entire file is considered unsafe to be used by more than one user. Sometimes (rarely), it is necessary to create a file that may never be accessed by more than one user at a time. This is done by setting the non-sharable (S) bit in the file's attribute byte. This can be done either as an option when the file is created, or later using the Attr utility. Once the non-sharable bit has been set, only one user may open the file at a time. If other users attempt to open the file, an error (#253) will be returned.

More commonly, a file will need to be declared non-sharable only during the execution of a specific program, for example, when the file is being sorted. This may be accomplished by opening the file with the non-sharable (S) bit set in the mode. If this is done, the file will be treated exactly as though it had the non-sharable attribute. If the file has already been opened by another process, an error (#253) will be returned.

A subtle feature of non-sharable files is that they may be duplicated using the I$Dup system call. This implies that they may be inherited, and therefore accessible to more than one process at a time. Non-sharable means only that the file may be opened once at a time. It is usually a very bad idea to have two processes actively using any disk file through the same (inherited) path, for other reasons anyway.

End of File Lock

A special case of record locking occurs when a user reads or writes data at the end of file. The user is said to have "EOF Lock", and will keep the end of file locked out until a read or write is performed that is not at the end of the file. This is the only case that a write call automatically causes any of the file to be locked out. It was done to avoid problems that could otherwise occur when two users want to simultaneously extend a file.

An interesting and useful side effect occurs when a program creates a file for sequential output. As soon as the file is created, EOF Lock is gained, and no other process will be able to "pass" the writer in processing the file. For example, if an assembly listing is redirected to a disk file, a spooler utility may open and begin listing the file before the assembler has written even the first line of output. Record locking will always keep the spooler 'one step behind' the assembler, making

the listing come out as desired.

DeadLock Detection

A deadly embrace, or deadlock, occurs (typically) when two processes try to gain control of two or more disk areas at the same time. If each process gets one area (locking out the other process), both processes will be stuck permanently waiting for a segment that can never become free. This situation is a general problem, not restricted to any particular record locking scheme or operating system.

When a deadlock is found by RBF, an error (#254) is returned to the process that caused it to be detected. It is easy to create programs that, when run together, generate lots of deadlock errors. The easiest effective way to avoid them is to access records of shared files in the same sequences in processes that may be run simultaneously. This happens naturally in most file access methods.

When a deadlock error does occur, it is not sufficient for a program to to re-try the operation "in error". If all processes used this strategy, none would ever succeed. It is necessary for at least one process to release it's control over a requested segment for any to proceed. This may also be accomplished by
aborting any of the deadlocked processes.

Specific Details for Particular I/O Functions

Open/Create

The most important rule to follow when opening files, is to not open a file for update if you only intend to read from it. Files open for read only will not cause records to be locked out, and generally help the system to run faster. If files are routinely opened for update on a multi-user system, users may often become record locked, sometimes for extended periods of time. When this occurs, users sometimes think the system has died, and exhibit panic behavior.

File permission checking occurs for all files named in the specified pathlist. This means that if you do not have permission to read a directory, you may not access any files in that directory.

The special "@" file should be used in update mode only with extreme care. To keep system overhead low, record locking routines only check for conflicts on paths open to the same file. The "@" file is considered different from any other file, and therefore will only recognize record lockouts with other users of the "@" file. Writing via the "@" file (to patch crashed disks, for example) should only be done in single-user mode. This file has been included as a convenience only; it is likely that problems will eventually occur if it is used in update mode regularly.

Read/ReadLine

Read and ReadLine cause records to be locked out only if the file is open in update mode. The locked out area includes all bytes starting with the current file pointer and extending for the number of bytes requested. Thus, if a ReadLine call is made for 256 bytes, exactly 256 bytes will be locked out, regardless of how many bytes are actually read before a carriage return is encountered. EOF Lock will be gained if the bytecount requested also includes the current end of file.

A segment will remain locked out until any of the following occur: another read is performed, a write is is performed, the file is closed, or a record lock SetStat is issued. Releasing a record does not normally release EOF Lock. Any read or write of zero bytes will release all locks on the file. This was included as a convenient way of removing locks.

Write/WriteLine

Write calls always release any record that is locked out. In addition, a write of zero bytes releases EOF Lock and File Lock if they have been gained. Writing usually does not lock out any portion of the file, unless it occurs at end of file, when it will gain EOF Lock.

Close

When RBF expands files, it does so in increments of at least the 'segment allocation size' (PD.SAS) sectors long. This means that usually more space is allocated than is required. When the file is closed, the excess space is usually trimmed off, and returned to free space. This strategy does not work very well for random-access data bases that expand frequently, by only a few records. What happens is the segment list rapidly fills up with small segments. A provision has been added to prevent this from being a problem.

If the file (open in write or update mode) is closed when it is not at end of file, the file will not be trimmed. In order to be effective, all programs that deal with the file in write or update mode must insure that they do not close the file while at end of file, or the file will lose any excess space it may have. The easiest way to insure this, is to issue a seek(0) before closing the file. This method was chosen since random access files will frequently be at some other place than end of file, and sequential files are almost always closed at end of file.

Seek

The seek call has no effect whatsoever on record locking, with the minor exception noted above in close. In particular, seek does not remove any record locks.

Makdir

Makdir creates its files in non-sharable mode. Since file attributes are checked at each pathlist element (see open/create), this means that makdir will return an error if it cannot gain non-sharable access to any directory specified. This can be a bit annoying sometimes, but it helps prevent certain recursive programs from getting out of control.

Del

Delete begins by trying to open the file for write access in non-sharable mode. If the file is already open, an error (#253) is returned, and the file is NOT deleted. All kinds of problems can and do occur when this is not enforced.

Set Status

Two new setstat codes have been added for the convenience of record locking. They are SS.Lock, for locking or releasing part of a file; and SS.Ticks, for setting the length of time a program is willing to wait for a locked record.

```
SS.Lock   (A) = path number
          (B) = SS.Lock code
        (X,U) = desired lockout size
```

This code locks out a section of the file from the current position for the number of bytes requested. If zero bytes are requested, all locks (Record Lock, EOF Lock, and File lock) are removed. IF (X,U) contains $FFFF FFFF, then the entire file is locked out, regardless of where the file pointer is. This is a special type of "File Lock", and remains in effect until released by SS.Lock(0) as described above, a read or write of zero bytes occurs, or the file is closed. There is no way to gain File Lock using only Read or Write system calls.

```
SS.Ticks  (A) = path number
          (B) = SS.Ticks code
          (X)=delay interval
```

Normally, if a read or write request is issued for a part of the file that is locked out by another user, RBF sleeps indefinitely until the conflict is removed. The SS.Ticks call may be used to cause an error (#252) to be returned to the user program if the conflict still exists after the specified number of ticks of the system clock have elapsed.

The delay interval is used directly as a parameter to RBF's conflict sleep request. This means that the value zero (RBF's default) causes a sleep forever for the record to be released. A delay value of one effectively means that if the lock is not released immediately, an error is returned.

Error Codes

Record Locking introduces three new RBF error codes:

E$Lock	$FC	#252	The desired segment is locked by another user.
E$Share	$FD	#253	This file is non-sharable and busy.
E$DeadLk	$FE	#254	The request would cause a deadlock if permitted.

The E$Lock error is normally never seen, since the default operation of RBF is to wait forever for the segment requested to become available. It should occur only if the SS.Ticks call has been made, and the tick count expired before the record became available. E$DeadLk is self explanatory, and should also be very infrequent. If it does occur, it usually indicates inconsistant file handling. The program(s) in question should be examined carefully to eliminate the error.

The E$Share error may occur for a variety of reasons, and is the only common record lock related error message that will be seen. The following conditions produce this error:

-Deleting a file that is open.
-Trying to open a non-sharable file that is in use.
-Trying to open a file that has gained "File Lock".
-Trying to open a file non-sharably that is in use.
-Makdir is unable to gain non-sharable access at some level.