P/N: 800-0345

SUN-1 SYSTEM REFERENCE MANUAL

Draft Version 1.0

July 27, 1982

The Sun-1 Workstation (TM Sun Microsystems Inc.) is a personal computer system combining high resolution graphics with powerful local processing and optional high speed networking. This document is intended to provide all information needed to install, operate, and program the Sun-1 Workstation.

This version completely supersedes all previous drafts. Your questions, corrections, and criticisms are welcomed. Please respond to Martin Rattner or Henry McGilton at Sun Microsystems, Inc.

Portions of this document are based upon material originally written at Stanford University by William Nowicki, Jeffrey Mogul, Tim Mann, Vaughan Pratt, and David Brown, whose contributions are gratefully acknowledged. Used with permission.

Sun Microsystems, Inc.
2310 Walsh Avenue
Santa Clara, CA 95051
(408) 748-9900

SUN-1 SYSTEM REFERENCE MANUAL

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUN-1 SYSTEM REFERENCE MANUAL

Draft Version 1.0

1. INTRODUCTION

1.1. Description

The Sun-1 Workstation (TM Sun Microsystems, Inc.) is a personal computer system that combines high resolution graphics with powerful local processing and optional high speed networking. The workstation is based on the Motorola 68000 processor. It has a 1024 by 800 pixel bit-map display, a fully up/down encoded keyboard, 256K bytes of on-board RAM memory with memory management, and a "RasterOp" mechanism for high-speed display updates.

The Sun-1 Workstation electronics consists of two PC boards: a processor board and a frame buffer (graphics) board. The workstation uses the Intel Multibus*, which is proposed IEEE standard 796, so many other common peripheral interfaces are commercially available.

Several optional interfaces are available for the Sun-1. An Ethernet interface card allows the Sun Workstation to connect to a 3Mb Ethernet local network. A disk interface allows up to four SMD disk drives to be connected, such as the Fujitsu M2313K or the CDC Lark**. A mouse pointing device may be connected to the existing parallel input port.

1.2. Physical Packaging

The Sun-1 Workstation consists of two physical units, a display module and a detached keyboard. The display module contains the video monitor, card cage, power supply, cooling fan, and back panel. The card cage has seven Multibus slots***, two of which are occupied by the processor board and the graphics board.

The back panel is illustrated in Figure BACKPANEL. The five connectors labelled "SMD" are used to connect to up to four SMD disk drives using the optional disk controller. Only the Sun keyboard should be plugged into the connector labelled "Keyboard". A mouse may be plugged into the "mouse" connector. The RS-232 serial ports labelled "A" and "B" are used to connect equipment such as terminals, modems, printers,

---

* Multibus is a trademark of Intel Corporation.
** Lark is a trademark of Control Data Corporation.
*** A small number of the earliest-manufactured units have six-slot card cages.

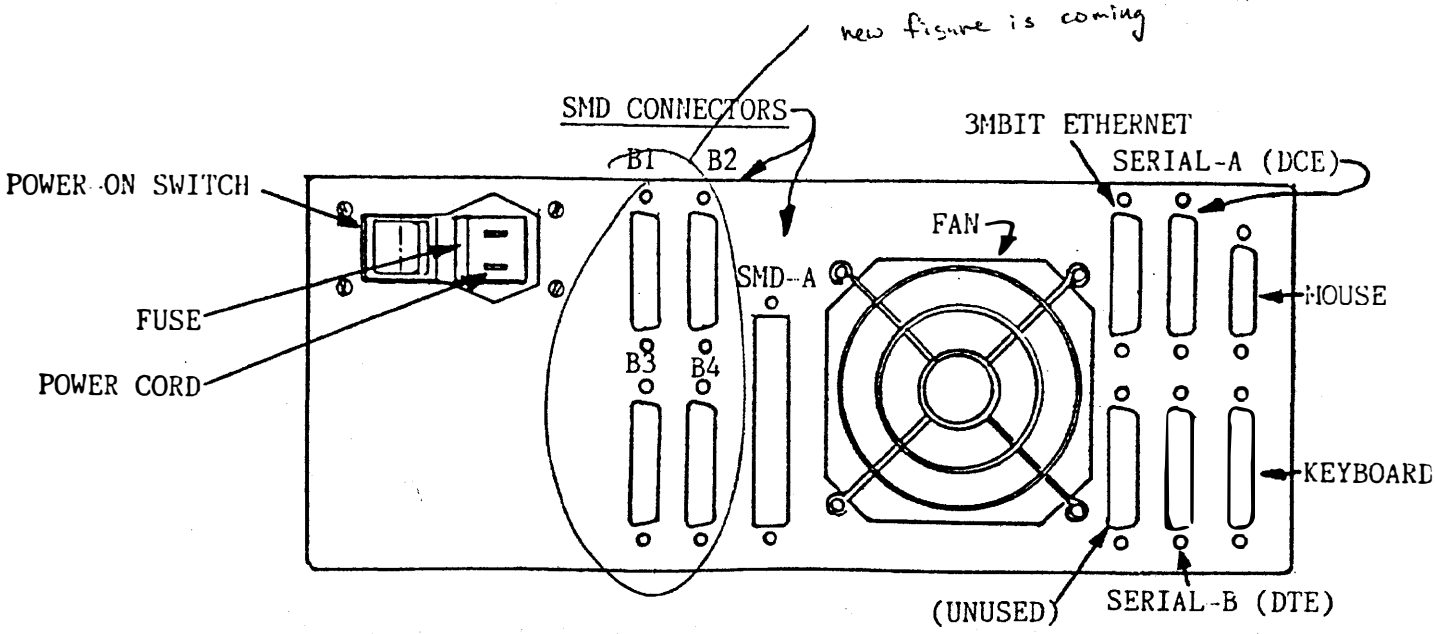


Figure BACKPANEL. The Sun Workstation Back Panel.

etc. See the chapter on "Installation" below for information about connecting peripheral equipment to the Sun Workstation.

Be sure to keep the area behind the fan unobstructed.

1.3. Notations Used in this Document

Integers are normally written in decimal; if preceded by "0x" they are hexadecimal.

Software interfaces and programming examples are specified in the C programming language*.

* For a detailed description of C, see Kernighan, B. W. and Ritchie, D. M., The C Programming Language, Prentice-Hall, 1978.

2. HARDWARE DESCRIPTION AND CONFIGURATION

2.1. Multibus Interface

The Sun-1 Workstation uses two busses: a synchronous, high-speed memory bus for processor-memory communication and the the IEEE 796 Bus (Intel Multibus) for input/output and peripherals. The IEEE 796 Bus is an asynchronous bus, accomodating devices with various transfer rates while maintaining maximum throughput. The Multibus provides 8-bit and 16-bit data transfers and 20-bit addressing, extendable to 24-bit addressing. It allows multiple masters to share the same bus, and provides serial (daisy-chain) as well as parallel priority resolution schemes. The Multibus has a secondary backplane connector, the P2 connector, for inter-module connections. This connector is used in the Sun workstation for the high-speed memory bus mentioned above.

In order to offer a consistent model for 68000 programmers, the Sun processor board generates 68000 byte order on the Multibus. This means that the low-order or the even byte is placed into bits D8 through D15, whereas the high-order or odd byte is placed into bits D0 through D7. If the processor board communicates with a byte-organized Multibus device, it is typically necessary to reverse the byte-order in software.

The Sun 68000 board generates the 20 address lines on the standard IEEE 796 Bus. Using these 20 address lines, the board can address up to one megabyte of memory and one megabyte of input/output locations.

The Sun processor board contains an on-board precision voltage reference for power-on reset. Multibus INIT is generated whenever the voltage falls below 4.5 Volt. In addition, Multibus INIT can be generated by executing the 68000 RESET instruction. Note that in this standard configuration the 68000 always drives INIT to the Multibus and the 68000 cannot be reset by an INIT from the Multibus.

If the 68000 accesses the Multibus and does not receive a data transfer acknowledge within 1 to 3 milliseconds, the access will be aborted via bus error. The timeout period in the Sun processor board includes the Multibus acquisition time. Thus, if a peripheral device locks up the bus for more than 1 millisecond, timeout can occur.

See also the section titled "Multibus Priority" below.

For more information about the Multibus, consult the following **references:**

R. W. Boberg, "796 Microprocessor Bus Standard", Computer, October, 1980.

This publication in the IEEE Computer Magazine gives a good introduction and overview of the bus.

IEEE, "Proposed 796-Microprocessor Bus Standard", with Errata, December, 1981.

This is the official copy of the Standard and includes up-to-date revisions and errata. It is available from:

MicroBar, Inc.
Attn: Rich Boberg
1120 San Antonio Rd.
Palo Alto, CA 94303
415-964-2862

Intel Corp., "Intel Multibus Specification", Revised April 1981.

This is the Intel version of the Standard, recommended in addition to the IEEE version. This document is a superset of the IEEE standard and contains some application-specific information. It is available as Manual Number 9800683-03 from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051
408-987-8080

or from local Intel sales offices.

Sun Microsystems, Inc., Sun 796-Bus Interface Manual, forthcoming.

This document will describe design considerations which apply when interfacing other Multibus boards to the Sun Workstation boards.

2.2. Processor Board

The Sun processor board contains a Motorola 68000 processor running with a 10 MHz clock, 256 Kbytes of RAM memory with byte parity, an Intel 8274 dual UART (equivalent to NEC7201), an Am9513 System Timing Controller, and a 16 bit parallel input port. For programming information on the UART and timer chips, see the manufacturers' data sheets*.

The next chapter on "Installation" provides instructions for installing or removing boards in the Sun-1 card cage.

* Component Data Catalog. Intel Corporation, Santa Clara, CA.; Am9500 Family Interface Manual, Advanced Micro Devices, Inc., Sunnyvale, CA.

2.2.1. UARTs

The Intel 8274 or NEC 7201 dual UARTs are wired as follows:

UART A is wired as a DCE port.

	on UART	on Connector
Outputs:	TXDA	RXDA
	RTSA	CTSA
Inputs:	RXDA	TXDA
	CTSA	RTSA
	DCDA	DTRA

Baud rate: generated by Timer 4.

UART B is wired as a DTE port.

Output:	TXDB	TXDB
Input:	RXDB	RXDB

Baud rate: generated by Timer 5.

2.2.2. Timers

The AMD 9513 timer chip is configured with an input frequency of 5 MHz and FOUT of 2.5 MHz (connected to Gate 1). It contains five timers, whose usage is described in the following table:

Timer	Usage	Operating Mode	Normal Frequency
1	Watchdog. OUT generates BERR/Reset. [1]	Out=TC, Repetitive	interval= 2.8 msec [2]
2	User timer. OUT causes INT6.		
3	Refresh timer. OUT causes INT7.	Out=toggle, repetitive (one shot). Reset by refresh routine.	interval= 2 msec [2]
4	UART A. OUT connected to UART A TX/RX Clk.	Out=toggle, repetitive.	16 times UART baud rate
5	UART B. OUT connected to UART B TX/RX Clk.	Out=toggle, repetitive.	16 times UART baud rate

Notes: [1] Causes reset if Refresh Timer Out (#3) is Low.
[2] Difference between timers 1 and 3 determines Multibus timeout period.

2.2.3. Multibus Priority

The Sun-1 processor board is always configured to be the highest-priority Multibus master, by grounding (asserting) Bus Priority In (BPRN) on the processor board.

If the 68000 board is used in conjuntion with a Multibus DMA board, such as a disk controller, that does not support Common Bus Request (CBRQ) then the 68000 board must be configured such that it gives up the Multibus after every Multibus cycle. This also will cause three additional wait states for each Multibus access. The Interphase 2180 disk controller is an example of a Multibus DMA board requiring this configuration.

On the other hand, if all Multibus DMA devices (Bus Masters) do support CBRQ, then the CBRQ jumper on the processor board is not required. Instead, the 68000 board will retain bus mastership until a lower priority master requests it by asserting CBRQ. Following a CBRQ, the current Bus Master has to yield mastership for at least one cycle.

The CBRQ jumper is the leftmost pair of pins at location J902 on the processor board, viewing the board with the Multibus at the bottom.

2.3. Graphics Board

The Sun graphics system is a high-resolution bit-mapped frame buffer and display processor on one Multibus board. The general organization of the graphics board is illustrated in Figure GRAPHICS. There is a small amount of hardware assistance to perform a set of simple high bandwidth operations called "RasterOps". This results in a simple yet flexible graphics device, with the performance needed to support sophisticated user interfaces. The section below entitled "The Sun Graphics System" contains detailed specifications of the programming interface to the Graphics board.

The graphics board has only one switch, which is illustrated in Figure GBOARD. This switch is used to set the Multibus base address (in Multibus Memory Space). It may be set to any address multiple of 0x20000 between 0x0 and 0xE0000.

The standard address for the graphics board is 0xC0000. Any additional graphics boards are placed at successively lower addresses.

The following table indicates which switch to set for each of the possible addresses. Note that ONLY the indicated switch should be turned on; all other switches should be in the off position.

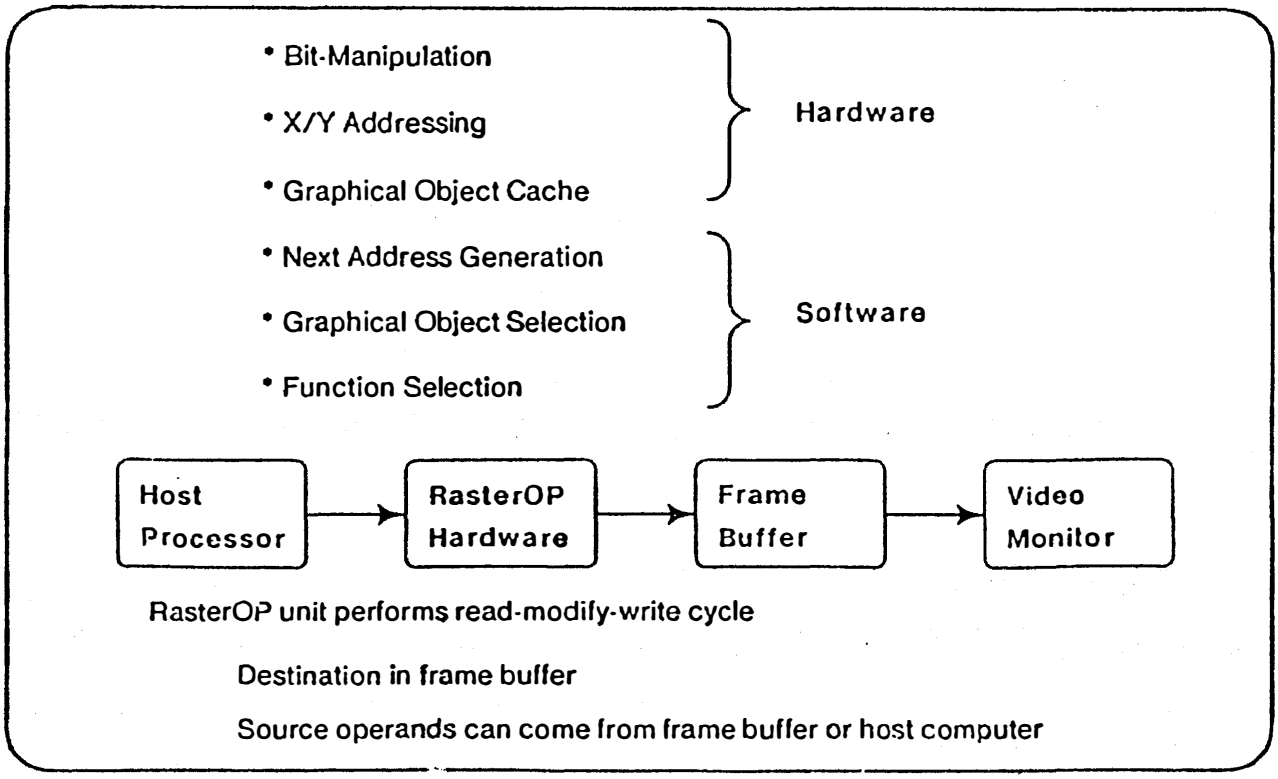


Figure GRAPHICS. The Sun Graphics Board.

address	DIP switch	
0x00000	8	
0x20000	7	
0x40000	6	
0x60000	5	
0x80000	4	
0xA0000	3	
0xC0000	2	(standard setting)
0xE0000	1	

2.4. Memory Expansion

The Sun memory expansion board provides 768K bytes of additional "on-board" memory for the Sun processor. It is connected to the processor board via the Multibus P2 connector and permits access by the 10MHz 68000 without wait states. Up to two of these board can be added to the Sun-1 workstation.

The memory expansion board has only one option: to be located starting at 256K or starting at 1M within the on-board memory address

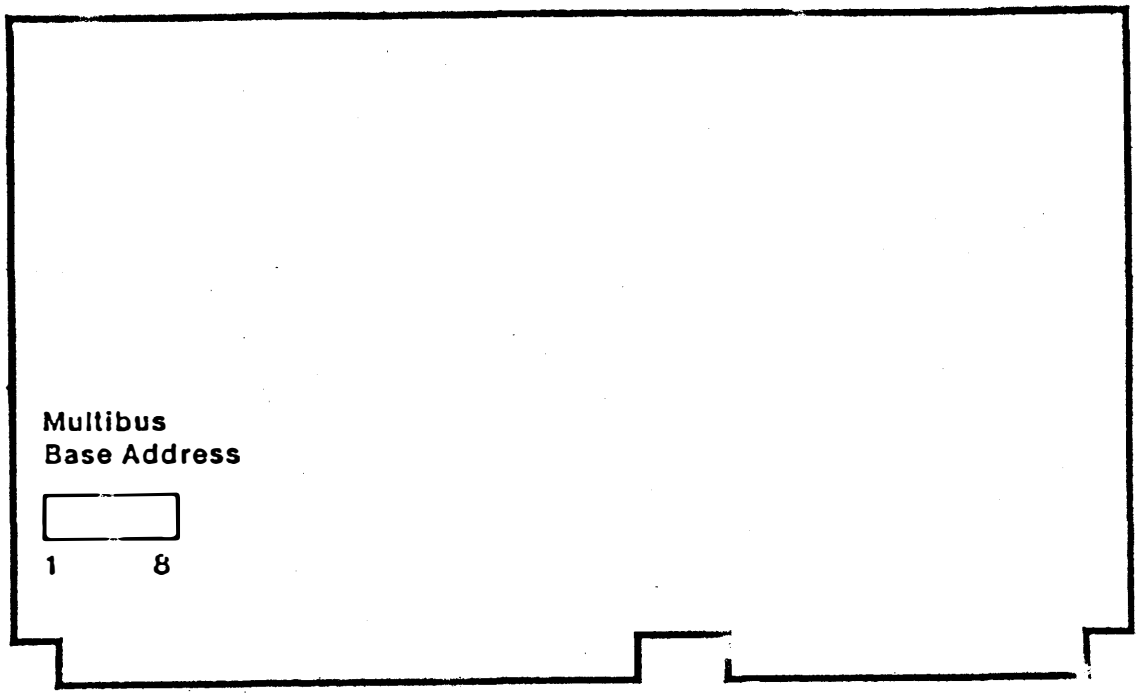


Figure GBOARD. Sun Graphics Board.

space. To configure the first memory expansion board to have a starting address of 256K, insert the 74S138 chip at location U1006. To configure the second board for a starting address of 1M, insert the 74S138 chip at location U1008. See the next chapter for installation instructions.

2.5. Video Display

The video display monitor currently supplied with the Sun-1 Workstation is the Ball Model HD17H CRT Data Display. This is a solid-state, raster-scan, high-density data terminal display with 17 inch diagonal screen size in a horizontal format. The service manual for this display is supplied with your workstation:

Service Manual, Ball CRT Data Displays HD Series, 5-017-1047.

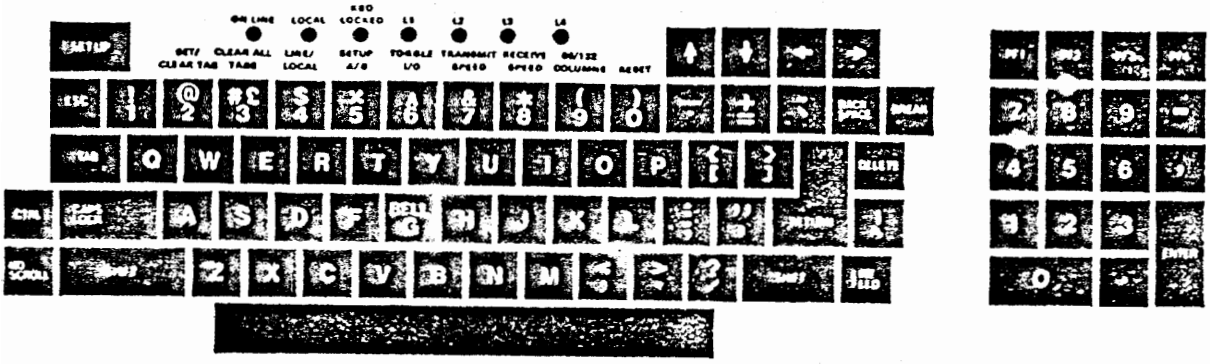
This manual provides full specifications and detailed operating and maintenance instructions for the monitor. Please heed the safety warnings in the manual. Service on the display should be performed only by qualified service personnel.

2.6. Keyboard

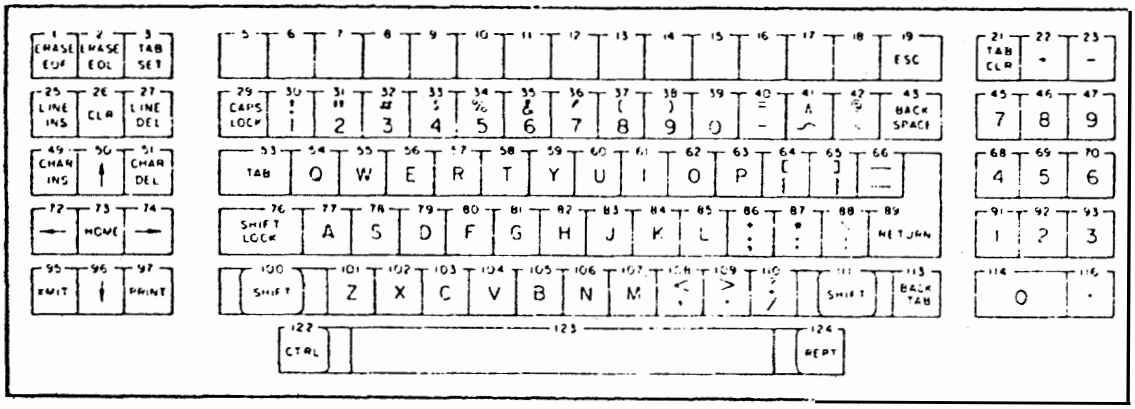
The Sun-1 Workstation is supplied with a detached keyboard available in two configurations (see figure KEYS):

(1) The Micro Switch 103SD30-2, a microcomputer-based Hall Effect keyboard which has been modified to produce full up/down keystroke encoding.

(2) The KeyTronic model P2441, a low-profile keyboard meeting German ergonomic requirements and featuring the DEC VT-100 key layout.



VT100 Keyboard



103SD30 Keyboard

Figure KEYS. Keyboard Layouts.

The keyboard cable attaches to main workstation unit via the back panel KEYBOARD connector (see figure BACKPANEL in section 1.2 above). See the next chapter, "Installation" for more information.

The key mappings are table-driven in software and can be redefined. The section below entitled "The Sun Keyboard" provides the details.

The keyboard is not user-serviceable. Refer servicing to Sun or an authorized service representative.

2.7. Ethernet Board

The Sun 3 MBit/sec Ethernet Board provides the connection of the Sun-1 Workstation to Ethernet-1, the experimental 3 MBit/sec Ethernet developed by Xerox PARC. The 3 MBit/sec Ethernet Board interfaces with the CPU via programmed I/O and interrupt. In Multibus notation, the board is an I/O slave with 16-bit addressing and 16-bit data paths. Note that the board is not readily compatible with 8-bit Multibus I/O.

The Ethernet board has two octal dip-switches: one to select the Multibus base address and one to select the local Ethernet host address. The location of these switches is shown in figure ETHERNET.

The Ethernet interface communicates with the host CPU via four read and four write registers located in Multibus I/O space. The registers are located on successive word (16-bit) boundaries starting on a 256-byte boundary within the 64K Multibus I/O space. Only the eight high-order address bits are decoded for the selection of the board. To select the Multibus base address, take address bits A8..A15 of the desired address and encode them into dip-switch S505. Switch #1 is the least significant bit, and "1" bits correspond to "on" switches. By convention, 0x100 is the normal address for the first Ethernet board, and subsequent boards (if any) are placed at successively higher addresses.

After obtaining an Ethernet host number from your local Ethernet administrator, express it in binary and set it into dip-switch S507. Switch #1 is the least significant bit, and "0" bits correspond to "on" switches, unlike the correspondence used for the Multibus base address. NOTE: Ethernet addresses 0 and 0377 (octal) are reserved for special Ethernet functions and should not be used as a host address.

Two separate documents, providing full specifications and installation instructions, are supplied with the Ethernet board:

SUN 3 MBit Ethernet Board

SUN 3 MBit Ethernet Board Installation Manual

Refer to these for further details.

2.8. Disk Controller

The Interphase SMD 2180 is an intelligent storage module controller/formatter, using bipolar microprocessor technology. It plugs directly into the Multibus and is a Bus Master during data transfers, using a variable burst length DMA technique. It directly connects via industry standard A and B cables to from one to four storage module drives which are available from a number of manufacturers. Sun Microsystems supports two such drives in particular: the Control Data Lark Micro Unit and the Fujitsu M2312K Microdisk Drive. Currently it is not possible to mix Lark and Fujitsu drives on the same controller board.

Coax Cable:

Type RG11/U Type Foam

Transceiver:

Xerox Part # 209926
TLC Part # 2000

Transceiver Cable:

Xerox Part # 216411D

Flatcable Assembly

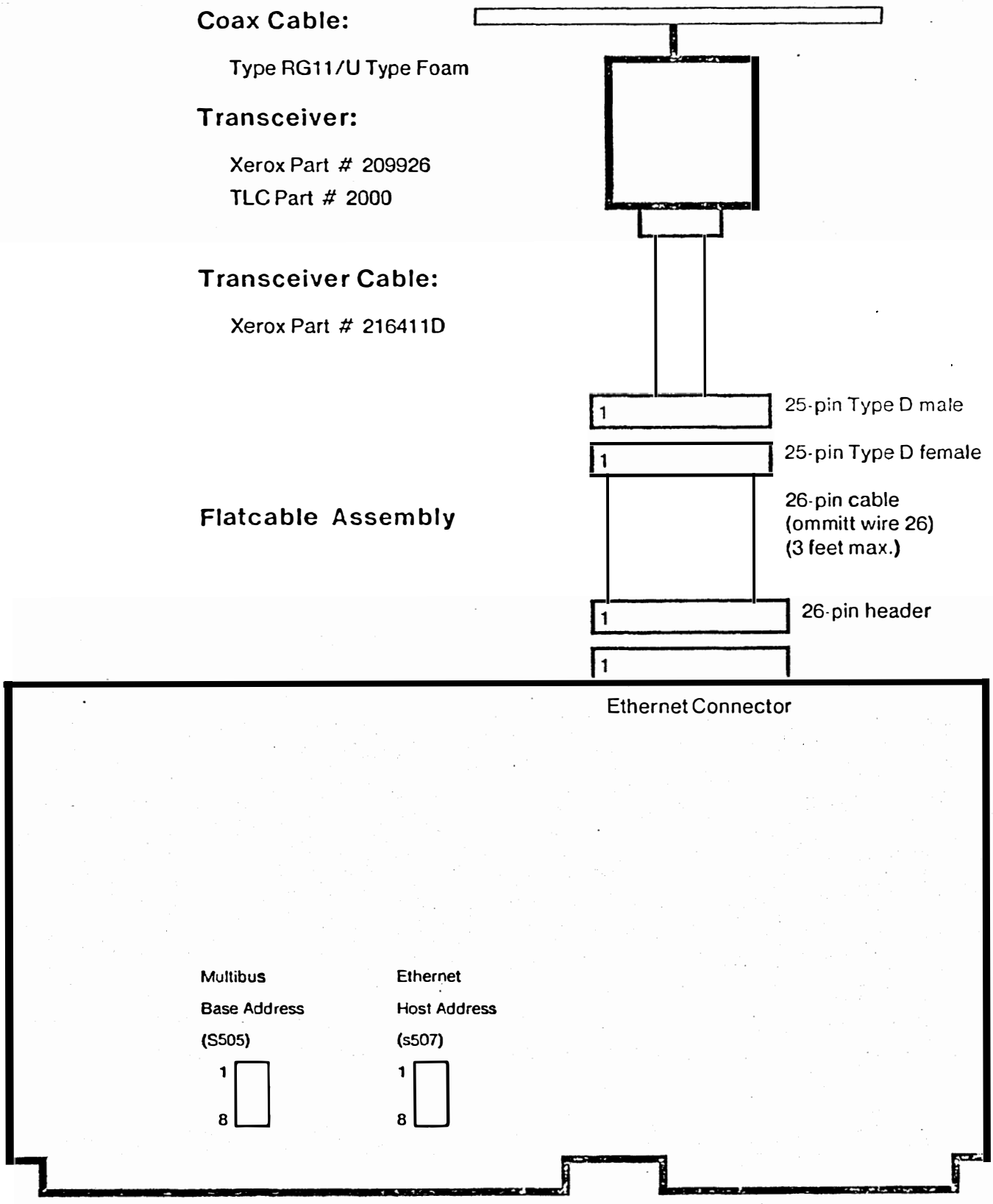


Figure ETHERNET. Sun 3 MBit/sec Ethernet installation.

You must install at least 64K bytes of Multibus memory in order to use the SMD 2180 disk controller.

Four sets of straps and two 8-bit dip-switches are provided for configuring the disk controller board. For UNIX* on the Sun-1, the M3 jumper must be installed and switch #5 on dip-switch S2 must be ON. Switches #7 and 8 on dip-switch S1 must be set as follows:

for Lark: Switches #7 and 8 ON.

for Fujitsu: Switch #8 ON, #7 OFF.

Switches #1-6 on S1 determine the address of the controller in Multibus I/O space:

Switch #5 ON, others OFF: address = 0x40.

Switch #5 and 1 ON, others OFF: address = 0x44.

Normally, the first controller is configured at 0x40, the next at 0x44. The controller(s) are normally configured to have lower Multibus priority than the 68000 board, by installing jumper E to F. See section 2.2.3 above for further explanation.

For the full specification and detailed description of the disk controller, see the supplied document:

SMD 2180 Storage Module Controller/Formatter User's Guide.

The next chapter on "Installation" provides instructions for installing or removing boards in the Sun-1 card cage.

2.9. Multibus Memory

The Chrislin CI-8086 dynamic RAM memory board interfaces to the Sun processor through the Multibus. Full specifications can be found in the supplied manual:

Memory System CI-8086 Technical Manual, Chrislin Industries

Each memory board is configured at a chosen range of Multibus memory addresses by placing a jumper at peg area B on the board and setting dip-switches SW1 and SW2. See layout drawing 70654 at the back of the Chrislin manual for the switch and jumper locations. Section 1.4 in the Chrislin manual specifies memory address selection.

For example, to configure a 128KB Chrislin board at Multibus memory locations 0 through 0x1FFFF, pegs B2-3 are jumpered and switches SW1-8 through SW1-5 are closed (set ON), with the remaining switches set OFF. The second 128K board would have jumper B2-3 and switches SW1-4 through SW1-1 set ON, for locations 0x20000 through 0x3FFFF. A third board

*UNIX is a Trademark of Bell Laboratories.

could be configured at 0x40000 by closing SW2-8 through SW2-5, and so forth. If you were using the 512K byte boards, all 16 dip-switches would be set ON; the first board would have jumper B2-3 (0x0 through 0x7FFFF) and the next board would have jumper B1-2 (0x80000 through 0xFFFFF).

2.10. Lark Disk Subsystem

The Lark is an 8-inch Winchester disk unit providing a maximum of 16.7 megabytes of unformatted capacity in the form of an 8.35 MB removable cartridge disk and an 8.35 MB non-removable disk. BEFORE UNPACKING, INSTALLING, OR OPERATING THE LARK DISK UNIT, PLEASE READ THE SUPPLIED MANUALS:

Control Data Lark (tm) Micro Unit Model 9454/9455, Volume 1 (Hardware Installation/Operation Manual)

Control Data Lark Power Supply and I/O Adapter (PIO), Volume 1 (Hardware Installation/Operation Manual)

Pay particular attention to the sections on installation and operation.

The Lark subsystem connects to the Sun-1 via the standard SMD A and B connectors. See the next chapter on "Installation".

2.11. Fujitsu Disk Subsystem

The Fujitsu M2312 is an 8-inch Winchester disk unit providing a maximum unformatted storage capacity of 84 megabytes. This unit contains non-removable disks in a sealed module. Since this is a sealed unit, there is nothing for the user to operate (other than the power-on switch) once the drive is properly installed and connected to the Sun workstation.

The Fujitsu subsystem connects to the Sun-1 via the standard SMD A and B connectors. See the next chapter on "Installation".

Detailed specifications and maintenance information is provided in the supplied Fujitsu manuals:

M2311K/M2312K Microdisk Drives CE Manual

M2311K/M2312K Power Supply CE Manual

Maintenance on the disk subsystems should be performed only by a qualified service technician.

3. INSTALLATION

3.1. Unpacking Instructions

Inspect all shipping cartons immediately upon receipt for evidence of damage. If any shipping carton is severely damaged, request that the carrier's agent be present when the carton is opened. If the carrier's agent is not present when a carton is opened and the contents are found to be damaged, keep all contents and packing materials for the agent's inspection.

Carefully unpack all items from the shipping containers. Avoid using sharp instruments which may damage the contents. We recommend that you save salvageable shipping cartons and packing material for future use in case the product must be reshipped.

3.2. Safety Precautions

Other than procedures described in this document, please do not attempt to service your Sun-1 workstation; contact Sun Microsystems or your field service organization.

Observe common-sense safety precautions as you would for any electrical or electronic equipment. Always disconnect power before opening any system enclosure. Whenever in doubt, contact Sun or an authorized service representative.

The following sections contain set-up and configuration specifications for the Sun-1 Workstation. Consult the additional hardware manuals supplied with your workstation (and with user-supplied Multibus accessory boards, if any) for further information and precautions.

3.3. Card Cage Configuration

The card cage slots are arranged as follows:

```
TOP
---
 1       Sun memory expansion 2
 2       Sun memory expansion 1
 3       Sun processor board
 4       Bus master 1
 5       Bus master 2
 6       Multibus memory, optional (graphics, other)
 7       Sun graphics board
---
BOTTOM
```

The earliest-produced Sun systems have a six-slot card cage. The arrangement of the six-slot cage is the same as above except that slot 6 contains the graphics board and slot 7 is deleted.

Note that the processor board and Sun memory expansion boards (if

any) MUST be located in slots 1, 2, or 3, since only these slots have the P2 edge connectors used by the Sun expansion memory.

"Bus Masters" are Multibus devices which initiate data transfers, such as the Sun processor board and the Interphase SMD 2180 disk controllers. Bus master(s), if present, must be placed in consecutive slots immediately below the processor board as shown in the diagram. In the Sun-1, the processor board is always configured as the highest-priority Multibus Master; see section 2.2.3 "Multibus Priority" above for more information.

A Multibus "Slave" is any Multibus device which does not initiate data transfers on the bus. This includes the Sun graphics board, the Sun 3Mbit Ethernet board, Multibus memory boards, serial multiplexer boards, etc. After bus masters and Sun memory expansion boards are positioned as required above, bus slaves may be placed in any empty slot. The graphics board is normally placed in the bottom slot to permit easier connection of cables.

3.4. Removal and Installation of Circuit Cards

Your Sun-1 Workstation is shipped with all cards properly installed and system tested. If you simply wish to set up and operate the Sun-1 in the configuration that was shipped to you, skip this section and proceed directly to the section "Set-up" below.

The following steps explain how to remove or install circuit cards in the Sun-1 Workstation card cage. To avoid damaging the Sun-1, this work is best performed by someone with prior "hands-on" experience. If you have any doubts about any step, please contact your service organization or Sun Microsystems for assistance.

(1) Disconnect the power cord to the Sun-1.

(2) Remove the six screws securing the tray, three on each side of the black SUN-1 pedestal.

(3) Carefully pull the tray out the back of the pedestal. Do not force it. Gently move cables out of the way as necessary.

(4) Before removing a card, carefully note the location and orientation of all cables attached to it. It may help to mark the flat ribbon cables with a felt-tipped marker. Note that, on most of the ribbon cables, pin #1 is distinguished by some marking, e.g. the red edge of the cable. This edge generally faces toward the front of the workstation at the point where the cable plugs into a header on the edge of a circuit card.

(5) Before removing any cards, it is necessary to remove the two black plates (know as "card cage restraints") at each end of the PCB tray. Remove any cables that are in the way, and extract or insert the desired card(s). To remove a card, simultaneously lift the two plastic levers provided for this purpose at each upper corner of the card. When inserting a card, make sure that it seats all the

way into the card cage. Pay attention to the position of the cards in the card cage (see "Card Cage Configuration" above). Observe that the Multibus edge connectors inside the card cage permit only one "right way" to install a card.

(6) If installing a user-supplied Multibus accessory card, follow manufacturer's instructions for connection of the card. It may be necessary to route cables around the top or side of the back panel. In some cases, you may have to leave the tray partially slid out or cut additional holes in the cabinet. Check with Sun Microsystems before modifying your workstation to make sure that you will not invalidate your warranty.

(7) Slide the processor card back into the Multibus card cage. Do not force; carefully fold internal cables into the enclosure. After you are satisfied that the installation is correct, replace the six screws securing the tray on the sides of the black display pedestal.

(8) To prepare the Sun-1 for operation, see the section "Set-up" below.

3.5. Internal Cabling

This section is not a complete specification, but provides some information on internal cable connections to assist you in properly reconnecting circuit boards. See the preceding section, "Removal and Installation of Circuit Boards". For more information on internal cable connections, see manufacturers' hardware documentation or consult a qualified service technician or Sun Microsystems.

The "headers" are connectors on the edge of some of the circuit boards, which are exposed when the boards are plugged into the Multibus card cage; i.e., they are on the edge of the board opposite the Multibus connector. Unlike some connectors, these headers allow their matching connectors to be inserted incorrectly "upside down", so you must take care to insert them correctly. To accomplish this, the plug should be inserted into the header so that pin #1 faces toward the front of the workstation, i.e., toward the video display. The plug itself may have a distinguishing marking toward one end; if so, this normally identifies pin #1. Also, the ribbon cables are usually marked in a highlighting color along one edge, indicating the pin #1 side. To be doubly sure you can replace a cable correctly, mark it for location and orientation with a felt-tipped pen before removing it.

The Sun graphics board has one edge-connected cable with six color-coded wires, connecting it to the (black & white) video display monitor. The side of the plug where most of the wires are connected is OPPOSITE to pin #1 and should face inboard, i.e. toward the back of the workstation. The pin #1 side of the plug is the side where no wires are connected and may also have a distinguishing mark on the plug itself; this side faces forward when plugged in correctly.

The processor board has two ~~16~~ 50-pin connectors (headers), one nearer the front of the workstation and the other nearer the back of the

workstation. See Figure BACKPANEL in section 1.2 for identification of the back panel connectors. The cables from back panel serial ports A and B merge into a single cable which plugs into the header (on the edge of the processor board) nearer the front of the workstation. Pin 1 of the connector, usually highlighted on the connector and/or indicated by the red edge of the ribbon cable, must face toward the front of the workstation at the point where it plugs into the processor board. The cables from the backpanel keyboard and mouse connectors merge into a single cable which plugs into the header nearer the back of the workstation. As above, the highlighted pin #1 side of the connector should be oriented toward the front of the workstation where it plugs into the processor board.

The SMD 2180 disk controller board connects internally to the back panel via industry standard A and B cables. The larger header nearest the front of the workstation is the A connector; this is connected directly to the back panel SMD A connector. As usual, the cable plugs into the header with the highlighted edge (pin #1) toward the front of the worstation. The four smaller connectors are the B connectors for up to four storage module drives. Cable number 1 is the farthest forward, i.e. closest to the A connector; if your workstation is configured for only one disk drive, the internal B cable for it will be plugged in there. Additional drives would take consecutive positions toward the rear of the controller board. As above, the B cables are inserted with pin #1 facing the front of the workstation.

The Sun 3 Mbit/sec Ethernet board has one cable connecting it to the back panel Ethernet connector. This connector, as above, should be oriented with pin #1 facing toward the front of the workstation.

3.6. Set-up

CAUTION: Before plugging in the power cord of any component of your Sun system, be sure that the supplied Volts and Hz are as specified on the back panel of your workstation. Use only three-prong (grounded) outlets. Always remove power before opening any system enclosure or servicing any system component. All servicing should be performed by qualified personnel.

The Sun-1 is supplied with a comprehensive set of ROM-based diagnostics. See Chapter 6, "Diagnostics", for details.

The back panels of the earliest-produced Sun-1 Workstations are unlabeled. Refer to Figure BACKPANEL in section 1.2 to identify back panel components.

3.6.1. Keyboard

The Sun keyboard should be plugged into the connector labeled "Keyboard" on the back panel. If you wish to use the Sun keyboard as your console input device (as is normally done) you must power-on the workstation AFTER plugging in the keyboard. See section 4.1 "Getting Started" for more details.

3.6.2. RS-232 Serial Ports

You may attach a terminal, modem, printer, plotter, or other device which interfaces through an RS-232 serial port, to one of the serial port connectors A or B on the back panel.

Note that serial port A provides the CTS, RTS, and DTR control lines in addition to the transmit and receive lines, while port B provides only transmit and receive. This may make line B unsuitable for connection to devices such as modems which require use of the control lines. Consult a hardware technician or Sun if assistance is needed.

Note also that Serial-A is a DCE port, which means that you can connect most terminals or printers directly to this port, while you probably need to interpose a "null modem"* if you wish to connect a modem or another computer. Serial-B, on the other hand, is a DTE port which permits direct connection of modems, computers, and the like (assuming, as noted above, that the control lines are not required) while requiring a null modem for attaching most terminals. Getting the cabling right is a problem which should be familiar to anyone who has had to connect RS-232 equipment; sometimes it is most easily solved by experimentation.

See section 2.2.1 "UARTs" for further specifications.

3.6.3. Disk Subsystem

Your Sun-1 may have been supplied with either of two optional disk subsystems: the Control Data Lark Module Drive Model 9455, or the Fujitsu Model M2312 disk drive. Both of these units use the industry standard SMD interface which is supported by the Interphase SMD 2180 disk controller supplied with your disk subsystem. It is possible to attach up to four disk drives to a single SMD 2180 controller; however, at present you cannot mix Lark and Fujitsu drives on the same controller.

The disk subsystem is attached to the Sun Workstation via two flat ribbon cables. The (wider) control or "A" cable plugs into the SMD-A connector on the Sun back panel; the (narrower) data or "B" cable for each of up to four drives plugs into one of the four SMD-B connectors on the back panel. If two disk controller boards are used, you will have to route the second set of A and B internal connecting cables around the top or side of the back panel to bring them outside of the enclosure.

The head assembly of the Fujitsu must be locked during shipment and should be locked any time the drive is to be moved, even from table to table. It must be unlocked before the drive can be used. To unlock it for use, open the disk subsystem enclosure by removing the screws on the sides, and follow the directions in section 3.5 of the Fujitsu Microdisk Drives CE Manual.

---

* A null modem is an RS-232 cable which reverses pins 2 and 3.

The Fujitsu subsystem is very simple to operate, having only a power cord and an on-off switch. We suggest that you take some time to look through the installation and operation sections of the Fujitsu manual before using the drive, to understand general operating requirements and precautions.

The Lark, with its front-panel controls and removable cartridge, is more complicated to operate than the Fujitsu. If you have the Lark, we urge you to read the set-up and operation sections of the Lark manuals supplied with your system before attempting to use your drive.

Like the Fujitsu, the Lark's head mechanism is locked for shipping and must be released before use. The manufacturer specifies that the carriage is to be locked any time the drive is to be moved, even from table to table. To release the lock, open the disk subsystem enclosure, and back off the screw labeled "carriage lock" until it is flush with the top of the top cover on the drive. The lock is set by turning off power to the drive and turning the locking screw in until it is observed to engage the carriage mechanism and a slight resistance is felt.

The main rules for operating the Lark are as follows:

(1) The drive cannot be operated unless a cartridge is installed.

(2) The cartridge loading door cannot be opened unless power is on AND the disks are "spun down" (at rest).

(3) With the power on and a cartridge installed, the disks are made to spin up by pressing the START-STOP button on the disk subsystem's front panel. When the button latches in, the disks will spin up. While they are spinning up, the green light on the START-STOP button will flash and the disks will not yet be usable by the Sun-1. When the green light burns steadily (a few minutes after pressing START-STOP) the disk is ready to use.

(4) The disks are made to spin down by pressing the start-stop button again to return it to the "out" position. The green light flashes while the disks are spinning down. The cartridge door cannot be opened until the disks are completely spun down, as indicated by the green light going out. This takes about a minute.

(5) When power is on and the disks are fully spun down, the cartridge door can be opened by squeezing upward on the latch button located in the top of the indentation in the door.

(6) After opening the door, you can eject the cartridge by pushing down firmly on the open door. Always keep the door closed when the drive is not in use. The manufacturer recommends keeping a spare unused cartridge in the drive when it is not in use.

(7) When loading a cartridge, be sure to insert it so that the arrows labeled UP/IN are facing up and pointing into the disk drive, respectively.

(8) To prevent the fixed disk from being written, push in the FIXED-PROT button on the front panel of the disk subsystem. The button will latch in and the red light will burn steadily. To write-enable the fixed disk, push the button again and the red light will extinguish. This button may be switched to the in or out position at any time.

(9) To write-protect the cartridge, move the small black toggle on the edge of the cartridge to the position labeled WRITE PROTECT. To allow the cartridge to be written, slide the toggle to the WRITE ENABLE position.

NOTE: You cannot change write protection on the cartridge while it is loaded in the drive. Make sure you have the switch set the way you want it BEFORE you load the cartridge, or you will be forced to wait through a spin-down/spin-up cycle to change it.

(10) If the Lark detects a fault condition (for example, you attempt to read the disk when it is not completely spun up) the red light in the FIXED-PROT button will flash on and off to alert you to the error condition. You can clear this signal by pressing the FIXED-PROT button twice.

Again, we urge you to read all of the installation and operation instructions in the Lark manuals before attempting to use the drive.

3.6.4. 3 Mbit/sec Ethernet Board

If the Sun 3 megabit per second Ethernet board was supplied with your system, consult the separate Ethernet documents supplied with the board for information on its installation and use. The Sun-1 connects to the Ethernet transceiver via the back panel Ethernet connector.

3.6.5. Mouse

The back panel Mouse connector provides a hardware interface compatible with currently used mouse devices. Software support will not be provided for this interface until the release of the forthcoming Sun mouse product. If you need to use this interface before then, contact Sun for assistance.

3.7. UNIX

If UNIX was supplied with your Sun-1 system, consult the accompanying document, "Installing and Operating UNIX on the Sun Workstation".

4. USING THE SUN PROCESSOR

4.1. Getting Started

After the Sun-1 Workstation has been properly installed (see the preceding chapter), the workstation and disk subsystem (if present) can be powered on. The "ON" position of the back panel power switch is to the left as you face the back panel.

After a few seconds, the monitor should identify itself on the console terminal, with a message looking like

Sun Workstation Monitor (Rev. C) - 0x100000 bytes of memory

If this message does not appear, and repeated use of the Power switch has no effect, make sure that power is being supplied from the outlet and that the Sun's fuse is not burned out. If you still have no success, consult chapter 6 for recommended diagnostic procedures, or contact your Sun Microsystems service organization.

If the console displays the message

Please clear keyboard to begin

above the Sun Monitor message, and the system doesn't respond to input from the keyboard, the likelihood is that one of the latching keys (CAPS LOCK or SHIFT LOCK) is latched in the down position, which prevents the keyboard from sending an idle signal to the monitor. Releasing all latched keys should solve the problem; if not, check the keyboard cable and connector to make sure that a proper connection exists. If the connection appears sound, try powering the workstation off and on again. If the problem persists, there is probably a defect in the keyboard or keyboard cable. Contact your service organization or Sun for assistance.

On the Sun keyboards, holding down the upper-left-most key (ERASE-EOF on the Micro Switch 103SD30) and typing an "a" causes a trap (also known as an "Abort") to the monitor so that debugging commands may be given. If the console device is an ASCII terminal connected to one of the UARTs (see the U command, described under "The ROM Monitor Commands" below), an abort is generated by pressing and releasing the "Break" key. You may continue an aborted program; see the C command, described below.

4.2. UNIX

If your Sun-1 is supplied with the UNIX operating system, the first thing you will typically do after powering-up the system is to initiate ("boot") UNIX using the monitor's Boot command. After booting, most of your interaction with the Sun-1 will be with UNIX rather than directly with the ROM monitor. To get started, see the accompanying document titled "Installing and Operating UNIX on the Sun Workstation". A complete reference on UNIX is provided in the UNIX Programmer's Manual, also supplied. After you have familiarized yourself with the features of UNIX on the Sun, we suggest that you return to this chapter and scan

through the remainder of this document, studying in detail only those sections that appear useful to you.

4.3. The ROM Monitor

Use of the Sun-1 Workstation involves interacting, at least initially, with the ROM-resident monitor. The remainder of this chapter discusses the purpose of the monitor, and how to use it.

Although the primary function of the ROM monitor is to provide a simple console for the workstation, there are a few features which affect the user programs that run under it. For simple programs, especially those using standard I/O routines, the characteristics of the monitor should not be important. However, if a program makes direct use of interrupts or I/O devices, a few critical details are relevant.

4.3.1. What is the monitor?

We will first give a brief description of the operation of the monitor, to provide a context for understanding the rules imposed upon user programs.

The monitor has four major functions: initialization on processor reset, memory refresh, terminal emulation, and "intelligent console" facilities. Although the last two may be the most visible, the first two are the most important; the processor would be essentially unusable without them.

The Sun processor may be "reset" in several ways:

- Power is turned on.
- By a console command (K1 or K2).
- When the "watchdog timer" detects that no memory refresh has occurred within 6 ms.

When the processor is reset, the monitor gains control. It initializes the on-board I/O devices (timers and UARTs), sizes memory, sets up the Segment Table and Page Table, initializes the on-board RAM to all ones, creates the RAM refresh routine, and initializes the interrupt and exception vectors and the monitor's RAM-resident global data. The monitor checks for the presence of a second PROM pair in the second set of CPU board PROM slots. The second PROM pair contains the Boot routine (invoked by the B monitor command) and a diagnostic routine which is automatically invoked on non-watchdog resets. After initialization, the monitor transfers control to a module that manages the "console" functions.

Memory refresh is done by the processor because it simplifies the hardware while not incurring any significant performance penalty. The memory is refreshed by simply reading 128 consecutive words every 2 milliseconds. The reads are done as instruction fetches by executing a routine consisting mostly of NOPs. This routine is stored in RAM, and so a malfunctioning program may damage it and thus cause havoc. The watchdog timer will detect this and reset the system within 6 ms. It

restores the refresh function and allows memory to be examined for diagnosis.

The console functions are implemented with fairly straightforward routines that communicate with the user via the Sun keyboard and display. If the keyboard is unavailable upon reset, the monitor will use UART A for input, which may be connected to a standard ASCII terminal. If the frame buffer (graphics board) is unavailable, UART A will be used for output as well. These default assignments of input and output device may be changed via the "U" command. See "The ROM Monitor Commands" below.

All monitor I/O is done using "busy-waits", and the code runs at the highest interrupt priority. Therefore, if a user program is interrupted by typing the Abort sequence on the console terminal or with some other exception, the monitor will run correctly unless its global data area has been damaged. If the user program is then continued, it should be unaffected by the interruption save for the possible loss of some console I/O interrupts.

4.3.2. Absolute Rules

The first 16K bytes of memory (addresses 0 through 0x3FFF) are reserved for the monitor and should never be written by user programs; however, user code may want to change exception vectors occasionally. It is legal to change any exception vector, except the following: the "Level 7 Autovector" at 0x7C (used for refresh timing), or any "User Interrupt Vector", between 0x100 and 0x3FF, inclusive. The refresh routine and monitor globals live in the region reserved for "User Interrupt Vectors", because the Sun processor board hardware does not permit their use as interrupt vectors.

Certain other exception vectors (Trace - 0x24, Trap #1 - 0x84 (Breakpoint Trap), and Trap #14 - 0xB8 (exit to monitor)) are used by the monitor. These may be altered without dire results, although the corresponding monitor facilities would not be available.

Any program altering the refresh routine or its interrupt vector must take responsibility for doing proper memory refresh and resetting the refresh and watchdog timers. See section "Watchdog Timer" below for more details.

4.4. The ROM Monitor Commands

The command format understood by the monitor is quite simple. It is:

<verb><space>*[<argument>]<return>

The <verb> part is always one alphabetic character; case does not matter. <Space>* means that any number of spaces is skipped here. <Argument> is normally a hexadecimal number or a single letter; again, case does not matter. Square brackets "[ ]" indicate that the argument portion may be optional. When typing commands, <backspace> and <delete>

(also called <rubout>, generated by the key labelled <back-tab> on the current Sun keyboard) erase one character; control-U erases the entire line. <Return> means that you should hit the carriage return key.

Several of the commands open a memory word, map register, or processor register. This causes the address or register name to be displayed along with its current contents. You may then type a new hexadecimal value, or simply <return> to go on the next address or register. Typing Q will get you back to command level. For registers, "next" means within the sequence D0-D7, A0-A6, SS, US, SR, PC. For example, the following commands set location 1234 to 5678, and register D1 to 0F00. The user types the underlined parts, with a return a the end of each command.

```
>e 1234
001234: 23CF? 5678
001236: 0000? q
>d
D0: 00000001?
D1: 00000231? 0f00
D2: 01203405? q
>
```

The commands are:

A n — Open A-register n (0 < n < 6, default 0). See the discussion above of "open".

B [dv(u,p)name] — Boot. Calls the boot routine, which is located at the beginning of the second pair of eproms on the processor board. Each of the arguments

```
     dv
     (u,p)
and     name
```

is optional; they default to "dk", "(0,2)", and "unix" respectively. For example, if you simply type "b", the command defaults to

```
B dk(0,2)unix
```

which is the standard UNIX boot. Similarly, if you type "b diag", the command defaults to

```
B dk(0,2)diag
```

The parameters dv, u, p, and name are passed to the boot routine, which interprets them as follows:

dv — A two-character string which identifies the boot device. Currently only "dk" (disk) is recognized. This will be extended for booting

from Ethernet, tape, etc. in future releases.

u Unit number, computed as (controller * 8) + (drive). Controllers 0, 1, 2, 3 are configured at Multibus I/O addresses 0x40, 0x44, 0x48, 0x4c. For example,
0 = drive 0 at Multibus I/O 0x40
8 = drive 0 at 0x44
9 = drive 1 at 0x44
etc.

p If <16, indicates partition number. If >16, denotes block offset.

name Name of boot file. The disk is assumed to be in UNIX format.

C addr Continue a program. The address addr, if given, is the address at which execution will begin. The registers will be restored to the values shown by the A, D, and R commands, except for the system stack pointer.

D n Open D-register n (0 < n < 7, default 0).

E addr Open the word at memory address addr; odd addresses are rounded down.

G [addr][param] Start the program by executing a subroutine call to the address addr if given, or else to the current PC. The values of the address and data registers are undefined; the status register will contain 0x2700. One parameter is passed to the subroutine; it is the address of the remainder of the command line following the last digit of addr.

K [number] If number is 0 (or not given), this does a "Soft Reset": it resets the monitor stack and the default escape character. This can be useful after exceptions or other anomalous situations. However, it may confuse the monitor if a breakpoint trap is set. If number is 1 this does a "Medium Reset", which re-initializes the page and segment maps without clearing memory. If number is 2, a hard reset is done and memory is cleared. This is equivalent to a power-on reset and causes the ROM-based diagnostics to be run (see chapter 6, "Diagnostics").

L Host-command This does an implicit U B, saving the current input and output device assignments. It then sends Host-command to the host computer, and sends a backslash character (hex 5c) to the computer to indicate that it is ready to be downloaded via the serial line. The Host-command should put the workstation back into normal mode after the file is downloaded by issuing an explicit U command,

which will restore the previous assignment of input and output device. For more details, see the section "Loading Programs" below.

M m — Opens Segment Map register m.

O addr — Opens the byte location specified. The byte vs. word distinction can be a problem on the Multibus, since some Multibus boards follow the 8086 convention for byte ordering within words, which is the reverse of the 68000 convention. See section 2.1 above for further details.

P p — Opens Page Map register p.

R — Opens the miscellaneous registers (in order) SS (Supervisor Stack Pointer), US (User Stack Pointer), SR (Status Register), and PC (Program counter). Alterations made to SS will have no effect.

S S-record — This causes the monitor to accept the S-record, described in section "S-record Format" below. Normally received from the host computer in L mode (see the L command above), this responds with a two-digit record count followed by a single letter, either L for length error, K for checksum error, or Y for success.

U [arg] — The U command manipulates the on-board UARTs and switches the current input or output device. The argument may have the following values ("{ab}" means that either "a" or "b" is specified):

{ab}	Select UART a (or b) as input and output device
{ab}io	Select UART a (or b) as input and output device
{ab}i	Select UART a (or b) for input only
{ab}o	Select UART a (or b) for output only
k	Select keyboard for input
ki	Select keyboard for input
s	Select screen for output
so	Select screen for output
ks, sk	Select keyboard for input and screen for output
{ab}#	Set speed of UART a (or b) to # (such as 1200, 9600, ...)
{ab}t	Enter transparent mode with UART a (or b)
t	Enter transparent mode with UART b
e	Echo input to output
ne	Don't echo input to output
x<char>	Set the transparent mode escape character to <char>; initially ^^ (hex 1e).
f	Use automatic flow control (^S/^Q) in transparent mode
nf	Don't use flow control

If no argument is specified, the U command reports the current values of the settings. If no UART is specified when changing speeds, the "current" input device is changed.

When received from the host computer in L mode (see the L command above), the U command causes the monitor to stop taking input from the host computer and restores the previous assignment of input and output device. In this case the argument, if present, is ignored.

Z [addr] Display or set the breakpoint. If addr is omitted, the breakpoint is displayed. If an address of zero is specified, the breakpoint is removed. Otherwise, the breakpoint is set to the given address.

4.5. Loading Programs

One of the primary uses of the monito is to load programs into the processor's memory. Programs can be loaded via a serial line connected to a host computer, referred to as "down-line loading".

4.5.1. S-record Format

Down-line loading involves transferring a program file over a serial line. The file must be converted into a format known as "S-records" before transmission.

In response to the L command, the monitor prepares to receive a sequence of S-records from the host computer, followed by a U command to return the monitor to interaction with the user.

An S-record is the standard Motorola EXORCISOR* download format. Each S-record has seven components:

1. The letter S.

2. A type, a single digit either 2 (signifying a Data record) or 8 (a Trailer record).

3. A two digit (one byte) count between 04 and FF, giving the total number of bytes in the address, data, and checksum (items 4, 5, and 6).

4. A three-byte address (six hex digits).

5. n-4 bytes of data, where n is the count given in 3. Each byte consists of two hex digits from 00 to FF.

* EXORCISOR is a trademark of Motorola Inc.

6. A one-byte (two digit) checksum. The checksum test is that the sum of the bytes in items 3 through 6 must be congruent to 255 mod 256, i.e. must have (hex) FF in the least significant byte.

7. The end of the line.

A complete download consists of a sequence of data (S2) records terminated by a trailer (S8) record. The trailer must appear. Each data record is loaded into memory starting with the address specified in the record, provided it passes the checksum test. The trailer serves two functions: to terminate loading, and to load PC with the trailer's address, giving a mechanism for defining the entry point of a program.

Consider the following sequence of four S-records:

```
S2080d3144190031f03b
S2080d31483310ca055f
S2080d314c0000112339
S8040d314A73
```

These four S-records load twelve bytes into memory starting at location 0xd3144. The starting PC is 0xd314A. The bytes which are loaded are:

```
190031f03310ca0500001123
```

4.5.2. Example of Down-line Loading

Suppose the file we want to load is called test.dl and the host command to download a file is "dlx <filename>". Assuming that you have used "transparent" mode to log into the host computer and initialize its command environment appropriately, you should then "escape" from transparent mode. Then, issue the command

```
L dlx test.dl
```

This will transmit the command "dlx test.dl" to the host, and then cause the monitor to accept subsequent commands from the host. The monitor sends a backslash character to the host when it is ready to begin receiving S-records. When the down-load is complete, the host should send a "U" monitor command, switching monitor input back to the console keyboard. You can then start your program with the G command. Normally, the current PC will have been set by the downloader to be the entry point of the program; if not, you can specify a starting address with G.

You can abort a download by hitting the "Break" key on the console terminal, changing to transparent mode via "U t", and interrupting the host. Down-line loading a file not in S-record format will probably cause strange behavior, therefore the host program should check the data it is downloading.

4.6. Traps

The monitor initializes the trap vectors so that it gets control of any exception or interrupt. Some, such as the memory refresh timer interrupt, are handled internally. Others have special meanings (for example, the "trap #1" operation is treated as a breakpoint trap). For exceptions or interrupts not internally handled, the monitor will print a message such as

Exception: Tr at <pc>

and then return to command level.

The messages printed use a two-letter code; here is a list of these codes and their meanings.

II Illegal Instruction: an illegal instruction code was executed

ZD Zero Divide: division by zero

Ch Check: a CHK instruction faulted

TV TRAPV: a TRAPV (trap on overflow) was taken

Pr Privilege violation: attempt made to execute privileged instruction while in user state

U0 Unimplemented 0: an opcode 1010 was executed

U1 Unimplemented 1: an opcode 1111 was executed

Un Unassigned: trap was made to unassigned vector.

L1, L2, L3, L4, L5, L6
Interrupt Autovector: an Autovectored interrupt was taken at one of levels 1 through 6.

Tr Trap: a trap instruction was executed.

Several exceptions are handled specially by the monitor. A breakpoint trap (instruction "trap #1") causes the message

Break at pc

to appear. A trace trap evokes the message

Trace trap at pc

to appear. Use of the keyboard abort sequence causes

Abort at pc

to appear. In each case, the pc shown is that of the next instruction to be executed. For further information on the use of these three

traps, see section 4.7, "Tracing Programs".

An Address Error trap (caused by attempting to access a word with an odd address) causes the monitor to print

Address Error: address <access-address> at <pc>.

There is a class of errors that may cause the processor to take a Bus Error exception. In the case of either an Address Error or a Bus Error, the <access-address> is useful in helping to determine the cause of the trap. Besides the access address, the monitor saves various information that may be useful for diagnosis. This is discussed in the next section, "Bus/Address Error Traps". It is possible to continue from these traps, although the apparent effect of the faulting instruction is not always predictable.

4.6.1. Bus/Address Error Traps

The monitor intercepts bus error exceptions and, by examining a wide variety of processor state components, classifies these errors according to the following scheme:

- Protection Error
  A reference was made to the <access-address> (see below) which is not allowed by the segment map for the access mode in effect at the time of the trap. For example, a program tried to write to a segment mapped read-only. (Note that the monitor always runs in supervisor mode, even after a trap from user mode.)

- System Space Error
  A reference was made in user mode outside of the space mappable by the segment map (i.e., the address was above 0x1FFFFF).

- Page Invalid Error
  A reference was made to a page which was not valid (that is, the page was not mapped to one of on-board memory, Multibus memory space, or Multibus I/O space.)

- Timeout Error
  A Multibus reference timed out; usually, this means that a device or memory was referenced but is not plugged into the card cage, or is not working right.

- Parity Error
  The wrong parity was seen on a read from on-board memory.

- Watchdog Error
  The watchdog timer expired and caused a bus error.

For any of these errors, the monitor prints a message such as

Protection Error: address <access-address> at <pc>

To assist the user in diagnosing bus error and address error exceptions, the monitor saves various processor state information at the beginning of its global data area. The address of this area is currently defined to be 0x200; however, the monitor will always store this address as a longword at location 0, so you will always be able to find the information by indirecting through location 0. You can examine this data with the "E" monitor command.

For more information on MC68000 exception handling, refer to the MC68000 User's Manual*.

The information saved by the monitor on a bus error or address error is described in the following table. On address errors, only the first three items (the first eight bytes) are saved. Following the 68000 conventions, bits are numbered with bit 0 the least significant. Words are 16 bits and longs are 32 bits.

Address	Size	Description
0x200	word	Misc. information saved by 68000: Bits 15-5: Not meaningful. 4: Was the access a read or write? (write=0, read=1) 3: Was the processor processing an instruction (=0) or not (=1)? 2-0: Function code. Possible values are 000, 011, 100 - Unassigned. 001 - User Data 010 - User Program 101 - Supervisor Data 110 - Supervisor Program 111 - Interrupt Acknowledge
0x202	long	Access address. The address which was being accessed by the aborted bus cycle.
0x206	word	First word of the instruction being processed.

The following information is saved only on bus errors:

0x208 through 0x247	long x 16	Saved contents of 68000 registers. The order is from data register 0 to data register 7, then from address register 0 through address register 7. A7 is the System Stack Pointer.
0x248	long	User Stack Pointer.
0x24C	long	Status Register, extended to a longword with high-order zeros.

* Section 5, "Exception Processing", MC68000: 16-Bit Microprocessor User's Manual, Third Edition, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.

0x250	long	Program Counter. The value saved is advanced an unpredictable amount, from two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error.

Note that the registers saved in locations 0x208 through 0x253 are in exactly the same order and format in which they are displayed when you type the "D" command to the monitor, then keep hitting RETURN.

4.6.2. Watchdog Timer

On the Sun processor board, one of the five on-board timers is connected so that if it ever goes off, the "Reset" line is asserted and the processor is restarted as if the power had just come on. The monitor attempts to prevent this by recycling the timer each time it goes through its memory refresh routine.

However, if the processor halts, or if the memory refresh is not done (because the interrupt vector or refresh routine was altered), this "watchdog" timer goes off and resets the processor. Otherwise, it would hang and its dynamic memory contents would be lost.

If the processor suddenly displays its restart message, you should assume that your program either halted or destroyed some monitor-specific data. It is also possible that a hardware failure is to blame.

Note that in many cases it is possible for the monitor to detect the difference between a true restart and a watchdog timer restart. It avoids clearing memory, and displays "Watchdog!" before the usual "Sun Workstation Monitor" message.

4.7. Tracing programs

The monitor provides several facilities for tracing program execution. They are quite primitive, however, and basically require you to understand your program at the machine code level. However, if you have a symbol table listing of your program, you will be able to at least know where each routine starts.

4.7.1. Breakpoint traps

The use of a Breakpoint trap (BPT) allows to run a program and regain control when execution reaches a certain location. The monitor currently can only maintain one breakpoint trap at a time. A breakpoint trap is set at address *x* by typing B x. The current breakpoint address may be queried by typing B alone. The current breakpoint may be cancelled by typing B 0.

You can then start or resume your program with the C (or possibly G) command. Execution will proceed until the trap is reached, at which point you will get a message such as

```
Break at 001000
```


At this point, you may examine registers, memory, etc. You may also clear the BPT or set a new one. You may continue using the C command, which will execute the "broken" instruction, and leave the breakpoint still set.

If you load a new program while a BPT is set, the monitor will normally be able to detect this. On the other hand, if you give the K1 command ("Medium Reset") while a BPT is set, and then set a new one, the monitor will be confused if the first trap is taken.

4.7.2. Trace traps

Support for Trace traps (single-stepping a user program) is minimal. To set a trace trap, you should use the R command, proceed to the Status Register (SR), and alter it so as to inclusive-OR it with 0x8000. Similarly, the trace trap can be cleared by ANDing the value of SR with 0x7FFF.

Once the trace bit is set in the SR, you should then give the C command to continue the program (the G command cannot be used, since it doesn't restore the SR); to start a program with the trace bit set, give the command C <starting-address>.

After each instruction is executed, the message

Trace Trap at <pc>

appears, where <pc> indicates the address of the next instruction to be executed. If you hit Return right after this message appears, the next instruction will be executed. If you enter any other commands, you must use the C command to continue.

5. PROGRAMMING THE SUN PROCESSOR

5.1. Processor

5.1.1. Physical Address Space

The Sun processor is provided with a map so that you can map pages of 2K bytes anywhere in your address space. The structure of a virtual address is described in the Memory Management section.

On power-up, the monitor maps the first megabyte of on-board RAM and memory expansion board RAM so that its physical and virtual addresses are identical. All segments, starting at segment 0, are fully mapped. Segments are initialized for all contexts identically. Segment protection is set so that both Supervisor and User modes have Read, Write, and Execute access to every segment. If there is less than 1 Mbyte of memory, all page map entries corresponding to nonexistent memory are invalidated.

Two other physical address spaces are mapped into the memory address space. Address from 0x100000 to 0x1EFFFF are mapped to Multibus memory space addresses 0 to 0EFFFF, respectively. The first 64K bytes of MultiBus I/O space is mapped at the top of the virtual address space, at addresses from 0x1F0000 to 0x1FFFFF. Most commercially available Multibus I/O devices use this space.

The logical address space of 24-bit addresses used by the programmer is divided into eight parts:

0x000000 - 0x1FFFFF Mapped address space, as described above. There are 256K bytes of on-board RAM and up to 2 memory expansion boards of 768K bytes each. This space can also be mapped into the Multibus I/O or Multibus Memory space.

0x200000 - 0x3FFFFF On board PROM0. See the discussion below on "boot state".

0x400000 - 0x5FFFFF On board PROM1.

0x600000 - 0x7FFFFF The on-board double UART. Channel A data register is at 600000, command register at 600002, Channel B data is at 600004, and B command is at 600006.

0x800000 - 0x9FFFFF On board Timer chip. 800000 is the Data register, 800002 is the Command register.

0xA00000 - 0xBFFFFF Page map. The page map entry used to map some virtual address X is addressed at virtual address X + 0xA00000. Note that the segment map entries for virtual address X should be set up before accessing the page map entries since the segment map entries determine which page map entries are accessed.

0xC00000 - 0xDFFFFF Segment map. The current value of the context register determines which group of segment map entries are addressed. The segment map entry used to map some virtual address X is addressed at virtual address X + 0xC00000.

When read as a short-word, the high-order 4 bits of any segment map entry give the current value of the context register.

0xE00000 - 0xFFFFFF Context register when written (the high 4 bits of the shortword.) When read, this returns the 20-bit value of the parallel input port; the monitor uses this input port for the keyboard and pointing device.

In "boot state", the state of the system after reset, read and execute accesses to any location 0x0zzzzz in mapped addresss space are redirected to come from the corresponding location 0x2zzzzz (in the PROM0 address space), but write accesses to the mapped address space go to on board RAM. Also, all interrupts (including normally "non-maskable" ones) are inhibited. In this way it is possible to initialize RAM just after reset. Boot state is exited by writing to the PROM0 address space.

5.1.2. Exception Handling

When a processor cycle can not be completed normally an exception is performed. Besides the exceptions caused by internal conditions, such as divide-by-0 or word-access to an odd-byte address, five external conditions can make it impossible to complete the current instruction or bus cycle. These external conditions which raise a Bus Error exception are: system space errors, segment map errors, page map errors, timeout errors, and parity errors.

System space errors are caused when a logical address greater than or equal to 0x200000 is accessed in user mode. These addresses are reserved for supervisor state to address the on-board system facilities. A segment map error indicates that the protection bits in the segment map did not allow the type of operation attempted. A page map error is caused by accessing an invalid page. Timeout errors occur for off-board references to the Multibus that are not acknowledged within one to three milliseconds. Most likely, nonexistent memory or a nonexistent device has been addressed. (There are no timeouts for on-board references because the on-board bus is synchronous and all cycles are always acknowledged.) Parity errors occur if a byte or word with odd parity is read from local RAM. Since parity can only be checked at the end of a memory read cycle, the 68000 cannot abort the cycle in which the error occured, but the next cycle.

Parity checking can be enabled or disabled under software control. When a write is done to PROM0 address space to exit boot state, the low order bit of the data written controls whether parity checking is done. If the low bit is a 1, parity checking is enabled, otherwise is is

disabled. Such a write may be done at any time (from supervisor state).

When a bus error occurs the cause of the error can be determined by checking whether the attempted access was to system space in user mode, whether a mapped access violated the segment protection code, or whether the page referenced was nonexistent. If none of the above caused the exception, then the exception was a timeout for bus accesses, or a parity error caused by local accesses in the previous memory cycle.

5.1.3. Interrupts

The 68000 has seven interrupt levels, numbered 1 through 7, with level 7 being the highest priority and level 1 the lowest priority. Interrupts are recognized for all priority levels greater the the current processor priority level contained in the 68000 status register. When an interrupt is acknowledged the processor priority is set to the level of the interrupt request.

A level 7 interrupt is special in that it is recognized even if the mask in the 68000's status register is set to 7, thus providing a non-maskable interrupt capability. A level 7 interrupt is acknowledged every time the interrupt request changes from a lower level to level 7, that is, level 7 interrupts are "edge-triggered".

The Multibus standard defines 8 interrupt lines, INT0 through INT7, with INT0 being the highest priority. Also, the standard recommends the interrupts be level triggered instead of edge-triggered to allow multiple interrupt sources on each interrupt line.

To avoid confusion for 68000 programmers, the numbering and the priorities of the interrupt lines on the Multibus were made to correspond to the definition of the 68000. Thus INT7 on the Multibus is the highest priority interrupt and INT1 the lowest. INT0 is not implemented. In addition, INT7 is non-maskable and edge-triggered, whereas all other interrupts are maskable and level-triggered.

Three interrupt lines are assigned to on-board interrupt sources: INT7 - Refresh Timer, INT6 - User Timer, INT5 - UART. INT7, INT6, and INT5 are ignored from the Multibus.

Interrupts are acknowledged by the 68000 in auto-vector mode, that is, the interrupt vector is generated internally by the 68000 and is not supplied by the device. Thus the INTA signal on the Multibus and the interrupt vector capabilities of the Multibus are not used.

5.1.4. Initialization

After hardware reset, the 68000 processor board comes up in a special "boot state". In this boot state the normal operation of the board is changed as follows:

1) The on-board PROM, normally residing in system address space, overlays RAM starting at location 0, and is also accessible under its normal location. Thus the initial program counter and stack

pointer are fetched from PROM at locations 0 to 3, whereas other bootstrap code can execute from normal PROM addresses.

2) Since the PROM is overlaid at location 0, read access to the on-board RAM and to the Multibus is disabled. However, write access is possible allowing initialization of exception and interrupt vectors in RAM.

3) All interrupts, including the non-maskable interrupt, are disabled in hardware. After leaving the boot state, non-maskable interrupts can occur at any time, and maskable interrupts can occur as soon as the interrupt mask in the status register is lowered to allow them.

Boot state is exited by writing once to any location in PROM0 address space. This write also enables or disables parity checking; see section "Exception Handling" above.

5.2. Memory Management

5.2.1. Overview

The Sun Memory Management Unit has been designed to support a multi-tasking operating system, such as Bell Labs' UNIX system. It provides address translation, protection, sharing, and memory allocation for multiple processes executing on the 68000. All accesses of the 68000 to on-board RAM memory, Multibus memory, and Multibus I/O space are translated and protected in an identical fashion. The Sun memory management provides all the necessary mechanism for demand paging and virtual memory and will be fully compatible with the 68010 virtual memory processor when it becomes available.

The memory management consists of a context register, a segment map, and a page map. Virtual addresses from the processor are translated into intermediate addresses by the segment map and then into physical addresses by the page map.

The page size is 2048 bytes, the segment size is 32K bytes (giving 16 pages per segment), and up to 16 contexts can be mapped concurrently. The maximum logical address space for a context is 1024 pages (2M bytes). The maximum physical address space that can be mapped simultaneously is 2M bytes.

The organization of the memory management system is shown in figure MEMMAN below. The addressing scheme used to access the segment and page maps, and the format of the segment and page map entries, is shown in figure MMADDR.

5.2.2. Context Register

In a multitask environment it is important to be able to switch between processes quickly without having to reload all the translation state information of a particular process. The context register is a 4 bit register which can be set under supervisor control to switch between 16 sections of the segment map. This permits 16 contexts to be mapped

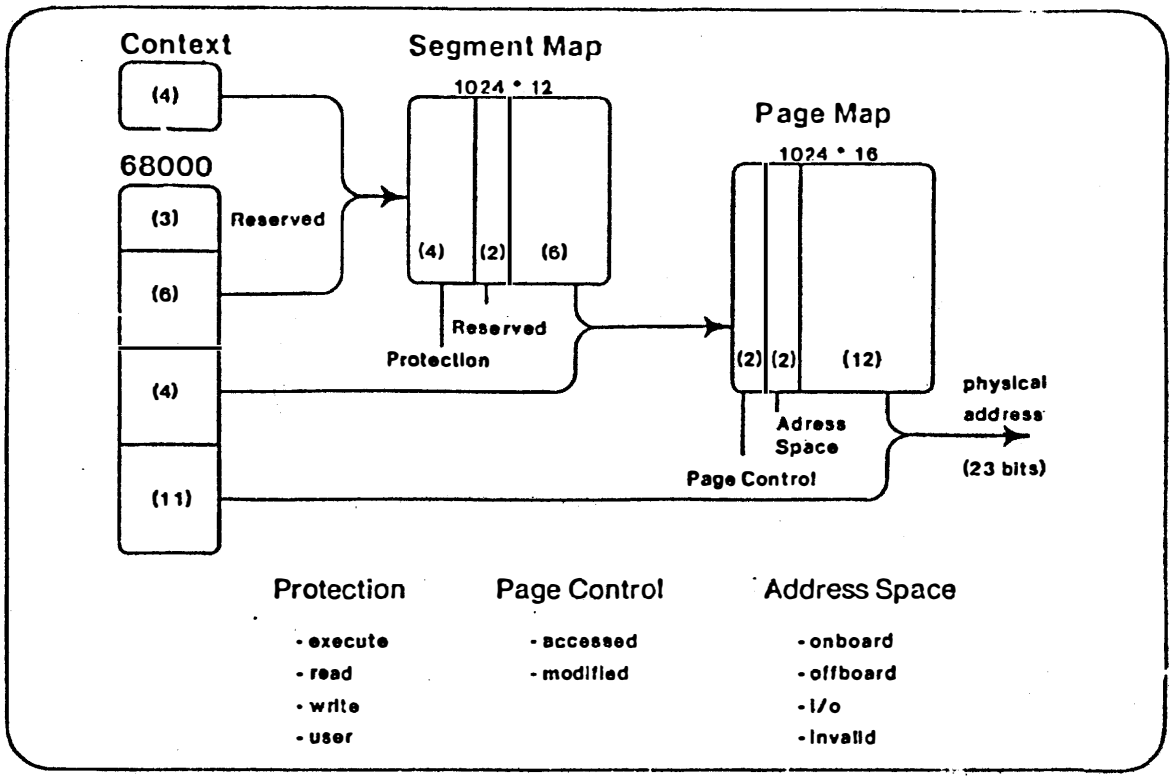


Figure MEMMAN. Memory Mapping on the Sun Processor.

concurrently; more than 16 contexts may be handled by treating the segment map as a context cache, replacing out-of-date contexts on a least-recently-used or other basis.

Each context has its own virtual address space. Sharing and inter-context communication may be implemented by writing the same values into the segment or page maps of multiple contexts.

A simple implementation of multiprocessing will allocate one context per process. More complex schemes are possible in which a team of processes occupies one context, or in which one process extends over more than one context (with context changes managed via system calls).

The context register is loaded by writing to location 0xE00000 with the desired value in the high 4 bits of the 16 bit data word. Note that the currently executing instruction stream at the time of a write must be mapped into both the old and new contexts at the same address (or must be in PROM). The context register is read by reading any of the segment map entries (starting at location 0xC00000) and examining the high 4 bits.

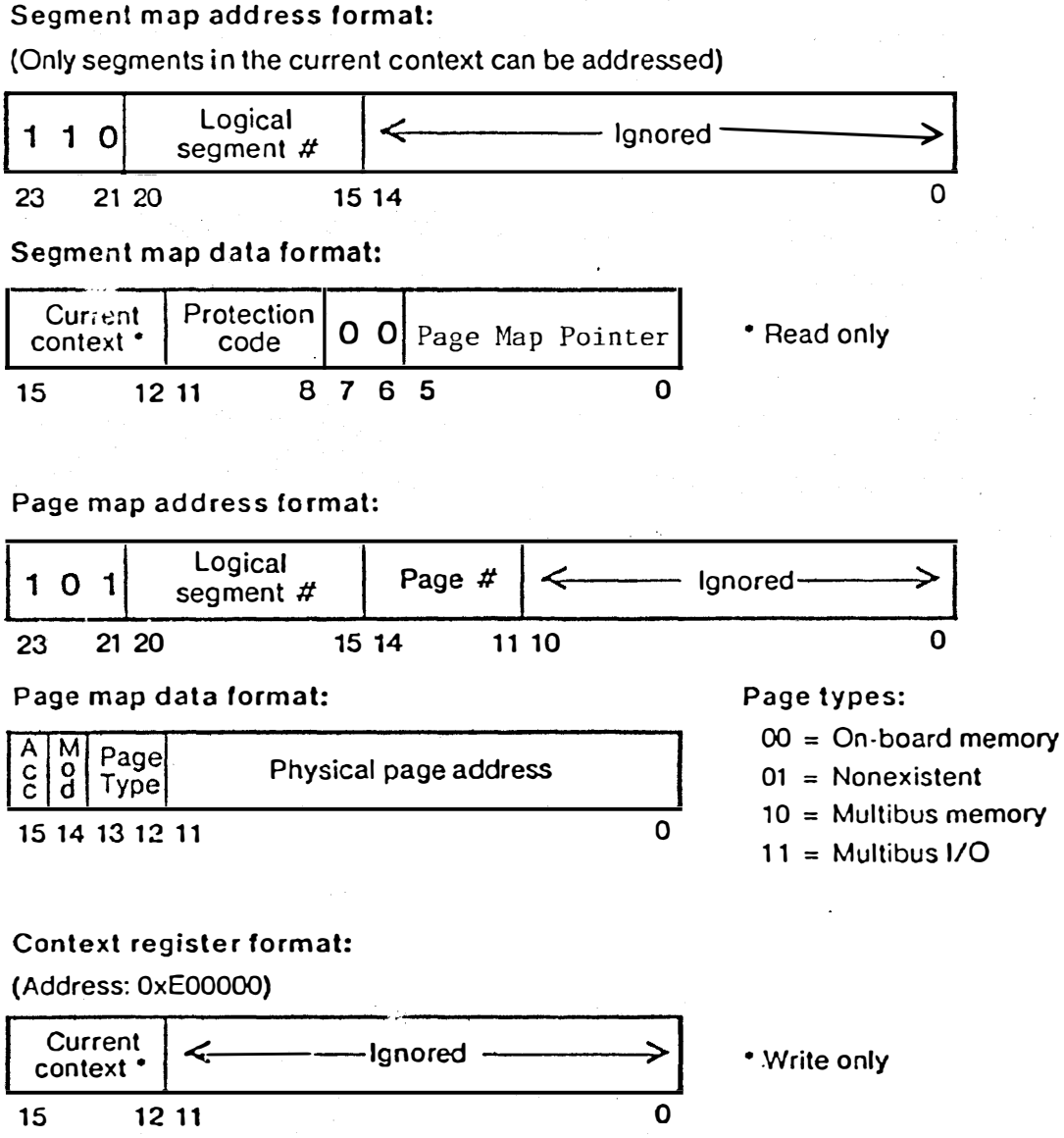


Figure MMADDR. Addressing Scheme for Segment and Page Map Entries.

5.2.3. Segment Map

The segment map has 1024 entries, indexed by the 6 most significant bits of the virtual address and the 4 bit context register. Thus, the segment map is divided into 16 sections of 64 entries, one section for each context. The segment map entry contains the 6 high-order bits of a pointer into the page map and a 4 bit protection code, defined below.

Only the 64 segments of the current context may be addressed at any one time. The current value of the context register determines which group of segment map entries are addressed. The segment map entry used

to map some virtual address X is addressed at virtual address X + 0xC00000.

Each virtual address space thus has 64 segments. Each segment can be mapped to 16 pages (32K bytes) or can be made inaccessible. The 16 page map entries pointed to by the segment map entry determine whether each 2K page exists and where it is located.

5.2.4. Protection

Protection is associated with the segment map; each segment has a 4-bit protection code. Since it is not possible to represent every combination of the six possible operations (user read, write, execute; supervisor read, write, execute) in four bits, 16 of the most useful combinations are provided. The 16 protection codes are defined in the following table. Full access is denoted "rwxrwx", with the first "rwx" being Read-Write-eXecute for the supervisor and the second for the user. A "-" denotes the absence of that privilege.

Code		Access by Supervisor	Access by User	Example of Use
0	------	None	None	Unused segment
1	--x---	Execute	None	System code
2	r-----	Read	None	System fixed data
3	r-x---	Read, execute	None	Mixed system code/data
4	rw----	Read, write	None	System variable data
5	rwx---	Full	None	Mixed system code/data
6	r--r--	Read	Read	User fixed data
7	rw-r--	Read, write	Read	System -> user transfer
8	r--rw-	Read	Read, write	User variable data
9	rw-rw-	Read, write	Read, write	System <-> user data
10	rw-r-x	Read, write	Read, execute	User-only code
11	rw-rwx	Read, write	Full	User-generated code
12	r-xr-x	Read, execute	Read, execute	Shared code
13	rwxr-x	Full	Read, execute	System-generated, shared
14	rwx--x	Full	Execute	Proprietary code
15	rwxrwx	Full	Full	Unprotected

5.2.5. Page Map

The page map handles the paging and the allocation of physical memory. A page map entry also indicates the physical address space in which a page is located, such as on-board or off-board memory. Further, the page map assists demand paging algorithms by maintaining reference and modified bits for each page.

The 6 bits from the segment map entry concatenated with the next 4 logical address bits from the 68000 form an index into the page map. Thus each segment accesses a block of 16 consecutive pages.

The output of the page map is 12 bits of physical address that is concatenated with the 11-bit byte address (the low 11 bits of the

virtual address) to form a 23-bit physical address. In addition, a page can be declared to be in on-board memory space, Multibus memory space, Multibus I/O space, or nonexistent, according to the following values of the page type field:

0 - on-board memory
1 - nonexistent
2 - Multibus memory
3 - Multibus I/O

A nonexistent entry indicates an invalid page, causing instruction abort. The address bits of such an entry are ignored and can be used by software.

Notice that each of the physical address spaces is 23 address bits (8M bytes) large. Since on-board memory is at most 256K bytes (1.75M bytes with two memory expansion boards) and the address space on the standard Multibus is at most 1M byte for memory and 64K bytes for I/O, some high order address bits of the page map entries will be ignored. It is the responsibility of the memory management software to provide correct table entries for a particular system configuration.

The page map entry used to map some virtual address X is addressed at virtual address X + 0xA00000. Note that the segment map entries for virtual address X should be set up before accessing the page map entries since the segment map entries determine which page map entries are accessed.

5.2.6. Page Control

In addition to the page mapping information, each page entry has two associated statistic bits, "modified" and "accessed", that are set whenever that page has been accessed or written into. These bits are updated automatically on all cycles for which access has been granted by the protection mechanism. Bit 14 of the page map entry is the "modified" bit and bit 15 is the "accessed" bit.

5.3. ROM Vector Table

The ROM monitor provides various services to user programs via the ROM Vector Table (VT). The VT is a convenient way of accessing monitor routines and data which does not depend directly on the absolute addresses used. A C-language program wishing to use the VT must include the header file "sunromvec.h" (see Appendix C).

The services provided by the monitor through the VT are available only to programs running in supervisor mode. Attempts to access addresses in ROM while in user mode will cause a hardware trap.

The following sections give more detailed descriptions of the variables and functions accessed through the VT, including C-language declarations. The C code to read a VT variable, for example OutSink, is:


```
*RomVecPtr -> v_OutSink
```

To store into a variable, for example EchoOn, code:

```
*RomVecPtr -> v_EchoOn = new_value;
```

To call a VT funtion, for example fwritechar, code:

```
(*RomVecPtr -> v_fwritechar)(char, *RomVecPtr -> v_FBAddr);
```

Note in this last example that the second argument to fwritechar, FBAddr, was also retrieved via the Vector Table.

5.3.1. VT Entries Used by the Hardware

`char *v_initsp;`
Initial System Stack Pointer loaded by hardware reset.

`int (*v_startmon)();`
Initial Program Counter loaded by hardware reset.

5.3.2. Information VT Entries

`char *v_SunRev;`
Revision level of monitor and hardware. This points to the string which is printed at power-up. The current revision level is Rev C.

`long *v_SerialNum;`
Serial number of this Sun. This is currently not implemented.

`long *v_MemorySize;`
Size in bytes of physical memory on the processor board and memory expansion boards. This does not include Multibus memory.

`GlobDes *v_GlobPtr;`
Points to the monitor's Global Ram structure, which contains all global variables used by the monitor. Various of these are also pointed to by specific entries in the Vector Table. This pointer should not be used without a good understanding of what it's pointing to...which is hard to obtain without listings of the monitor source.

5.3.3. Single-Character I/O

`unsigned char (*v_getchar)();`
Returns the next character from the current input source. If v_EchoOn is set (see below), the character is also echoed on the current output sink using putchar(). The input is done using busy-waiting; that is, if there is no input bufferred up, getchar waits until a character

arrives. Currently, serial ports can buffer up to 3 input characters (in hardware), and the console keyboard can buffer up to 30 keys (in software).

```
int (*v_putchar)(c);
char c;
```

Prints the specified character on the current output sink. If c is a linefeed ("newline"), it is preceded by a carriage return. The output is done using busy-waiting; that is, if the output sink is a serial port, and it cannot accept a character to transmit, putchar() waits until it can. Output to the frame buffer always occurs immediately.

```
int (*v_mayget)();
```

Attempts to return the next character from the current input source. If a character is pending, the result is in the range 0 - 255. Returns -1 if no character is pending. mayget() does not echo, regardless of the setting of EchoOn.

```
int (*v_mayput)(c)
char c;
```

Attempts to put a character to the current output sink. Returns 0 if it did or -1 if it didn't (because the current output sink was a serial line, and it was busy).

```
unsigned char *v_EchoOn;
```

getchar() echoes its input if and only if this variable is true (non-zero).

```
unsigned char *v_InSource;
```

Selects current input source:

0 - Sun keyboard
1 - serial port A
2 - serial port B

```
unsigned char *v_OutSink;
```

Selects current output sink:

0 - Sun frame buffer
1 - serial port A
2 - serial port B

5.3.4. Sun Keyboard Input

These Vector Table entries provide access to the monitor's support for the Sun Microsystems parallel keyboard. The next section, "The Sun Keyboard", describes keyboard operation in greater detail. Appendix D contains C-language listings of the keyboard header file and translation tables.


```
int (*v_getkey)();
```
If no key has been struck, returns NOKEY (defined as -1). Otherwise, returns either the up/down key code or a byte of ASCII for the next key struck, depending on the setting of v_translation (see below).

```
int (*v_InitGetkey)();
```
Initializes the polled side of the parallel keyboard interface. Must be called once before getkey is first called. Called by the monitor after a reset, so user programs will typically not need to call it.

```
unsigned int *v_translation;
```
The setting of this variable determines what kind of translation will be performed by getkey(). If translation = TR_NONE (defined as 0) then getkey returns "raw" up/down key codes. If translation = TR_ASCII (defined as 1) then keystrokes are translated to ASCII characters. These are the only translation options currently defined.

```
unsigned char *v_KeybId;
```
Identifying byte supplied by keyboard [not yet implemented].

```
keyboard **v_CurKeyboard;
```
Address of the current keyboard translation tables. See Appendix D for a listing of the current tables. The tables are in ROM, but the pointer to them is in RAM, and may be modified if you construct your own tables. If you do, see SetTable() and keyentry() below.

```
keymap * (*v_SetTable)(shiftmask);
unsigned int shiftmask;
```
This routine returns the address of the keymap in effect for the current keycode, based on which shift keys are currently depressed and whether the current key is moving up or down. Shiftmask is a bit vector which encodes the current position of "shift-like" keys and the direction of the current keystroke. See the header file "keyboard.h" in Appendix D.

```
int (*v_keyentry)(entry, keycode);
unsigned char entry, keycode;
```
This function is called when an entry in the keyboard translation tables is found which is not implemented by ROM code. If you modify the keyboard tables and need to add new kinds of entries, this "hook" lets you implement them when the key is struck. Note that this is called for the OOPS and HOLE entries even though they have defined meanings, since the ROM code does nothing when presented with them.

The parameters are the entry from the translation tables, and the keycode which represents the key depression or

release. Access to other assorted variables of the keyboard translation code is possible thru v_GlobPtr, which is described above under "Information VT entries".

If keyentry returns NOKEY, the mainline code of getkey() ignores this key depression and looks for the next key. If it returns any other integer, that integer is returned as the result of getkey().

keybuftype *v_Keybuf;
The circular buffer which holds up/down key codes. This is filled at memory refresh time (every 2 ms or so) as key codes are received from the keyboard, and emptied asynchronously by calls to getkey().

5.3.5. Frame Buffer Output and Terminal Emulation

These Vector Table entries provide access to the monitor's terminal emulation routines, which allow the frame buffer to be written to as an ANSI/4014 terminal. The section "Frame Buffer Terminal Emulation" below describes this interface in greater detail.

```
int (*v_finit)(G);
struct dpyglobals *G;
```
Frame buffer initialization routine. G points to the terminal emulator's global workspace. Two entries in the workspace must be filled; the first longword must contain the virtual address of the frame buffer, and the second must contain the virtual address of a 4096-byte work area. finit() fills the work area with its character font definition, which is stored in compressed form in the ROM. Typically, callers just use the workspace used by the monitor, which is accessible in the Vector Table as v_FBAddr (see below). The monitor calls finit() after a reset, so if a frame buffer exists at the usual physical address, typical user programs will not need to call finit().

```
int (*v_fwritechar)(c,G);
char c;
struct dpyglobals *G;
```
Writes the character c to the frame buffer "terminal". G points to the terminal emulator's global workspace; the address of the workspace used by the monitor is accessible as v_FBAddr in the Vector Table (see below). The monitor implements an ANSI standard (X3.64) compatible terminal in software. It also includes vector-drawing capabilities similar to the Tektronix 4014. Details of the escape code sequences supported are in the section "Frame Buffer Terminal Emulation" below.

long *v_FBAddr;
Virtual address of the frame buffer. This resides at the beginning of the terminal emulation global workspace, hence FBAddr can be passed as the "G" parameter to routines

requiring the address of the global workspace (v_finit, v_fwritechar, and v_fwrstr). If this value is changed, you must call finit() before any subsequent calls to fwritechar(), fwritestr(), putchar(), or mayput().

```
unsigned short (*v_FontTable)[][];
```
Address of the font table, which contains the character bit maps. This is filled in by finit() from a compressed font stored in ROM. It may be subsequently altered by users, if they know what they are doing. See the section "Frame Buffer Terminal Emulation" below for details.

```
int (*v_fwrstr)(addr,len,G);
unsigned char *addr;
short len;
struct dpyglobals *G;
```
Writes a string of characters to the frame buffer "terminal", more efficiently than a sequence of calls to fwritechar. Addr points to a character string of length len. G points to the terminal emulator's global workspace; the address of the workspace used by the monitor is accessible in the Vector Table as v_FBAddr (see above).

5.3.6. Mouse Support

The monitor does not currently support the optional mouse. There is one entry in the Vector Table for it, however. This entry, v_MouseBuf, is currently always zero.

5.3.7. Operating System Support

These Vector Table entries provide services which an operating system would find useful.

```
(*v_SetSeg)(Context, MapIndex, Value);
long Context, MapIndex;
short Value;
```
Sets an entry in the segment map. To provide optimal hardware performance, the Sun processor board does not allow a segment map entry for one context to be accessed while executing in another context. However, a routine which executes from ROM does not depend on the current setting of the context register and can perform this service for a RAM-resident operating system. This procedure sets segment map entry MapIndex (between 0 and 63) in context Context (between 0 and 15) to value Value (16 bits).

```
short (*v_GetSeg)(Context, MapIndex);
long Context, MapIndex;
```
Gets an entry from the segment map. To provide optimal hardware performance, the Sun processor board does not allow a segment map entry for one context to be accessed

while executing in anothe context. However, a routine which executes from ROM does not depend on the current setting of the context register and can perform this service for a RAM-resident operating system. This procedure reads segment map entry MapIndex (between 0 and 63) in context Context (between 0 and 15) and returns it.

(*v_KeyFrsh)();

This contains the address of the monitor's refresh interrupt service routine. This address is normally contained in the hardware vector pointer for interrupt level 7 (nonmaskable interrupt) at location 0x7C. If an operating system or standalone program wants to change or augment the refresh routine, it should set the hardware vector to point to its code, and have its code jump to KeyFrsh() when done. If an operating system or standalone program wants to change the refresh routine, it may restore the hardware vector from here when it wants to change back. Note that this entry, while declared as a function, is NOT a C function and cannot be called as one. It is called by interrupt hardware, and returns with the "rte" instruction instead of the subroutine's "rts".

int (*v_AbortEnt)();

This contains the address within the monitor of its "Abort" entry point. If an interrupt routine (such as a replacement refresh routine) wishes to allow the user to suspend the current program and enter the monitor, it can enter at this location. The routine must enter by a "jmp" to AbortEnt. Upon entry, all registers must have the same values they had on entry to the interrupt routine, and there must be no extraneous data on the supervisor stack (beyond the SR and PC stacked by the interrupt). Once entered by AbortEnt, the monitor will print an "Abort at xxxxxx" message and accept commands from the user. If a "c" command is entered, it will restore all the registers to their values on entry at AbortEnt, then restore the PC and SR from what was on the stack on entry at AbortEnt, resuming the mainline program. The monitor's refresh routine uses AbortEnt if the "Abort sequence" (Erase EOF then A) is typed on the console keyboard, or if a serial line is selected as current input source and its BREAK key is pressed.

int *v_RefrCnt

This integer value contains a millisecond counter which is incremented by the monitor's refresh routine. Since the refresh routine runs every 2 milliseconds, the counter is always incremented by 2. Due to uncommon conditions resulting from Multibus accesses to nonexistent memory or devices, this counter will run slightly "slow", so it is not suitable for a wall-clock value. It can be used for low-precision timing, though; for example, it is used to implement auto-repeat for the console keyboard.


```
char * (*v_bedecode)(beinfo, sr);
struct BusErrInfo *beinfo;
short sr;
```

Decodes a bus error and returns a pointer to a string which describes the cause of the error. The first argument points to an 8-byte area which contains the information stacked by a bus error. The second argument is the status register value stacked by the bus error. The result points to one of the strings "Spurious Bus", "System Space", "Protection", "Page Invalid", "Parity", or "Timeout".

Since a bus error can also occur if the refresh routine is damaged, bedecode() also restores the refresh routine to its original condition, and refreshes memory. If the refresh vector pointer at location 0x7C was invalid, it prints the message "xxxxxxxx=bad refresh vector" on the current output device, where xxxxxxxx is the previous contents of location 0x7C. In cases where the bus error was caused by the watchdog timer detecting loss of refresh, this will prevent the timer from resetting the system.

5.3.8. Interfaces Between Proms

These Vector Table entries permit routines in the second prom set (U102 and U104) to access common command-parsing and simple output formatting routines in the first prom set. They will not be documented here other than to list their names: linbuf, lineptr, linesize, getline, getone, peekchar, gethexbyte, getnum, message, printhex, diagret.

5.4. The Sun Keyboard

The console keyboard currently supplied with the Sun-1 Workstation is a Micro Switch 103SD30 Series with an 8048/8748 on the board. This has been modified to produce up/down keycodes on 8 parallel output lines. Each keycode is held stable on these lines for a minimum of 2.5 ms in order that the main processor can read it during its refresh routine, which executes every 2 ms or so.

When no keys are depressed, the keyboard transmits a keycode of IDLEKEY (defined as 7F hex) to indicate that that is the case. Thus, when the last key is released, a keycode for its release is sent, followed by an IDLEKEY. [In forthcoming versions the 8048 will, when idle, alternate its IDLEKEYs with keyboard id codes which identify the key layout or other factors which host programs might want to determine automatically. This feature is not yet implemented.]

User programs read from the keyboard by calling getkey() via the ROM vector table (see the preceding section, "ROM Vector Table"). By default, getkey returns ASCII codes; it can be persuaded to return key up/down codes by modifying the variable translation, also accessible thru the Vector Table.

Figure KEYBOARD shows the Micro Switch 103SD30 keyboard layout. The number above each key is that key's "down" keycode; the "up" keycode

is the same number plus 128.

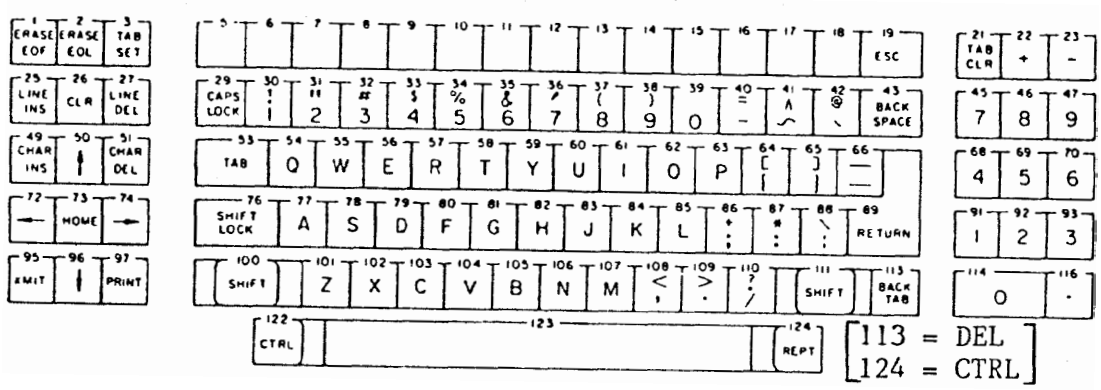


Figure KEYBOARD. The Sun Keyboard.

In ASCII translation, the keyboard operates like a normal terminal keyboard. It generates ASCII codes which corresponds to the keycaps except for the following keys:

REPT
(Key #124) This key is a second CTRL key, not a repeat key.

BACK TAB
(Key #113) This is a DEL key, generating 0x7F (Delete or Rubout).

ERASE EOF, ERASE EOL, TAB SET, TAB CLR, LINE INS, LINE DEL, CHAR INS, CHAR DEL
(Keys #1, 2, 3, 21, 25, 27, 49, 51) Not implemented. These keys are ignored.

CLR
(Key #26) Generates ^L (FF, Form Feed, 0xC).

unlabeled function keys
(Keys #5-18) These keys are ignored.

The codes generated by the keyboard are modified by several shift and shift-like keys. These are:

left and right SHIFT
(Keys #100, 111) Cause the uppercase letter or symbol to be generated.

SHIFT LOCK
(Key #76) Same as SHIFT keys. This key is unlocked by pressing either SHIFT key.

CAPS LOCK

(Key #29) Causes letter keys to generate uppercase letters; no other keys are affected.

left and right CTRL (right CTRL labeled "REPT")

(Keys #122, 124) Causes control characters to be generated. See table below.

There is one special sequence which can be typed to generate an "Abort", which causes the currently executing program to be interrupted, allowing commands to be typed to the monitor. This is generated by pressing the ERASE EOF key, then, without releasing it or pressing or releasing any other keys, pressing the A key. If the monitor's refresh routine is running. this will cause the message "Abort at xxxxxx" (where xxxxxx is the program counter value from the mainline program that was interrupted) and the monitor prompt character ">". To resume, type "c".

Note: When the monitor is using UART A or B for the console input, Abort is caused by a BREAK.

Special keys: the arrow and HOME keys generate ANSI standard control sequences (ESC [A, B, C, D, and H).

The following table gives the keystrokes necessary to generate the ASCII control characters (hex 0 through 1F). The notation CTRL-x means hold down the CTRL key (like a shift) and strike "x". Where more than one keystroke entry is listed, any of the alternatives will produce the given control character. Note that the SHIFT, SHIFT-LOCK, and CAPS-LOCK keys have no effect on control characters; i.e., CTRL-a is equivalent to CTRL-A, CTRL-~ is equivalent to CTRL-^, and so on.

Sun Keyboard Control Characters

Hex value	ASCII name	Keystrokes:
0	NUL	CTRL-space, CTRL-@
1	SOH	CTRL-A
2	STX	CTRL-B
3	ETX	CTRL-C
4	EOT	CTRL-D
5	ENQ	CTRL-E
6	ACQ	CTRL-F
7	BEL	CTRL-G
8	BS	BACKSPACE, CTRL-H
9	HT	TAB, CTRL-I
a	LF	CTRL-J
b	VT	CTRL-K
c	FF	CLR, CTRL-L
d	CR	RETURN, CTRL-M
e	SO	CTRL-N
f	SI	CTRL-O
10	DLE	CTRL-P
11	DC1	CTRL-Q

12	DC2	CTRL-R
13	DC3	CTRL-S
14	DC4	CTRL-T
15	NAK	CTRL-U
16	SYN	CTRL-V
17	ETB	CTRL-W
18	CAN	CTRL-X
19	EM	CTRL-Y
1a	SUB	CTRL-Z
1b	ESC	ESC, CTRL-[
1c	FS	CTRL-\
1d	GS	CTRL-]
1e	RS	CTRL-^
1f	US	CTRL-_

5.5. The Sun Graphics System

5.5.1. The Frame Buffer

The Sun graphics hardware consists of a bit mapped memory, coupled with a Raster-function unit capable of performing bit manipulation functions on the contents of this memory. The graphical memory is one megabit, corresponding to a graphical region of 1024 by 1024 picture elements (pixels). Each pixel has one bit resolution, i.e., is either ON or OFF. The term "frame buffer" as used in Sun documents refers either to the graphics board or to its bit-mapped memory; where the meaning is not clear from context, a more specific term is used.

Although there are 1024 by 1024 pixels in the frame buffer, only a region 800 pixels high by 1024 pixels wide is displayed on the video screen, as shown in Figure SCREEN. The pixels not displayed (1024 by 224) are still accessible, and may be used as a cache to store bit maps which are not visible but which may be moved (or copied) into the visible region by one of the raster operations described below.

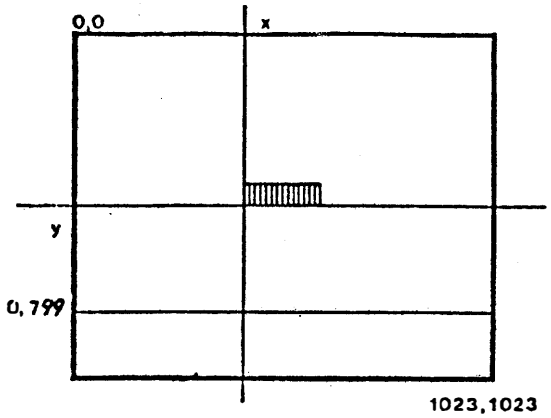
The pixels in the frame buffer are addressed via <X,Y> coordinates (column and row), with the upper left hand corner of the display corresponding to <0,0>. Positive X displacement is to the right; positive Y displacement is downward. The frame buffer allows the access of up to sixteen horizontally adjacent pixels on one read or write cycle (one access to the graphical memory). This yields increased bandwidth when a contiguous group of pixels is operated on by the same function.

The details of frame buffer operation are described below.

5.5.2. RasterOps

Figure RASTEROP illustrates the concept of a raster operation or "RasterOp", as developed by Newman and Sproull [1]. A RasterOp sets a

[1] Newman, William M. and Sproull, Robert F., Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill, 1979.



Size:	1024 * 1024 pixels
Visible:	1024 * 800 pixels
Invisible:	1024 * 224 pixels
Updat s:	16 pixels/cycle

Figure SCREEN. The Sun Graphics Screen.

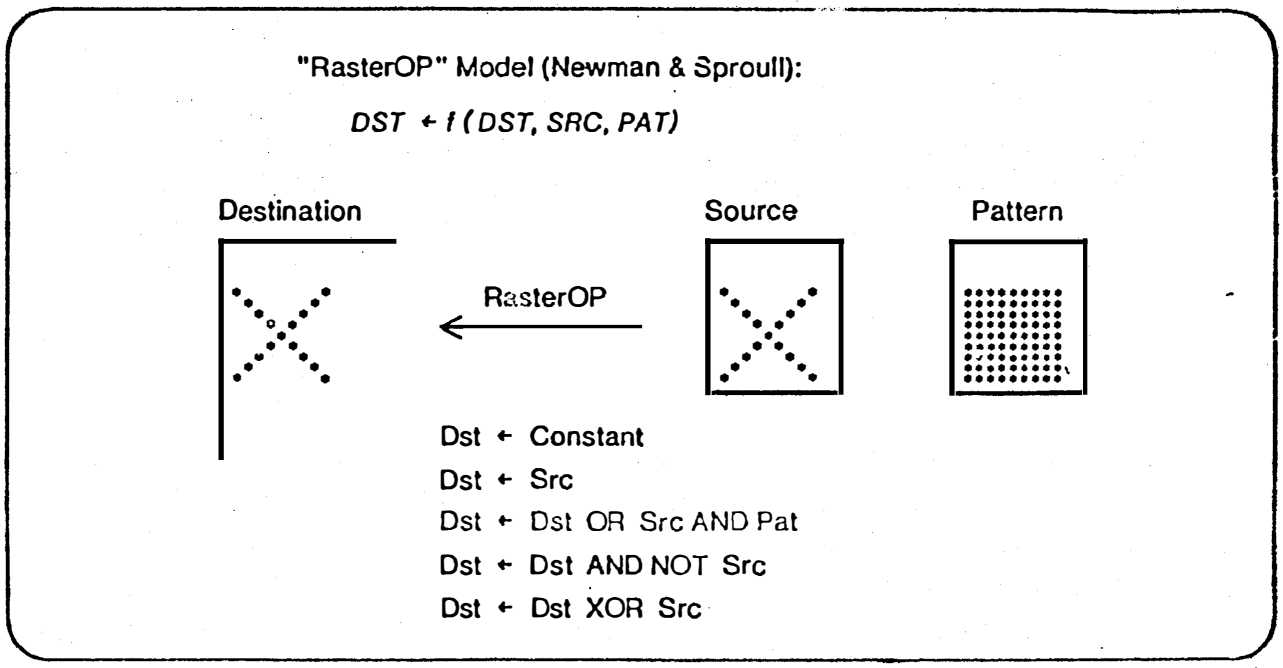


Figure RASTEROP. The "RasterOp" Concept.

destination rectangle on the screen to a bit-by-bit boolean function of three variables: its original contents (Dst), a source rectangle (Src), and a repeating bit pattern (Pat). There are 256 possible functions mapping three boolean operands into a boolean result. The frame

buffer's eight-bit FUNCTION register selects one of these at a time by acting as a three-bits-in, one-bit-out lookup table for corresponding bits of the Destination, Source, and Pattern. For example, suppose we want to set Destination equal to (Dst OR Src), ignoring the value of the pattern. Consider the application of this function to a single pixel. The function may be expressed in tabular form as shown in figure RASTEREXAMPLE.

Pat	Src	Dst	result
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

result = Src OR Dst

Figure RASTEREXAMPLE. A Boolean function.

The Pat, Src, and Dst columns in the table form an index running from zero (000) through seven (111). The eight bits of the result column uniquely specify the desired boolean function, and these are precisely the eight bits which are to be loaded into the frame buffer's FUNCTION register. By convention, the least significant bit of the function appears at the top of the table, hence this function (Src OR Dst) is represented by the eight-bit value (hex) EE. Examples of other function encodings are 0 (clear destination bits), FF (set destination bits), and CC (copy source to destination).

The Sun graphics system allows all 256 possible RasterOp functions, although only a few are used in practice.

For example, to clear the entire screen, the constant function 0 is applied to the viewable rectangle. To flash a certain window, the function NOT Dst is performed on that window. To write a character, the Src function is used, while NOT Src writes the character inverted (black on white), Dst OR Src overwrites (paints) the character, and Src OR Pat writes the character with a background pattern.

During application of a RasterOp, the Destination, Source, and Pattern are held in the frame buffer's Destnation Register, Source Register, and Mask Register, respectively.

5.5.3. Frame Buffer Addressing

The frame buffer is a dual ported memory, providing storage for a 1024 by 1024 pixel bitmap image. One port of the frame buffer connects (thru the function unit) to the host processor; the other port is

dedicated for video-refresh, with priority given to the video refresh. The processor port can access the frame buffer once every microsecond for a 16-bit operation.

The frame buffer is addressed in a cartesian coordinate system, in which <0,0> is the upper-left corner of the screen. From one to 16 horizontal pixels can be read or written in a single cycle, starting at the current <x,y> position (regardless of its byte alignment), with the data bits within a word being left-justified. For example, a 4-bit update at location <200,300> will write the most significant four data bits (D15 through D12) into locations <200,300> through <203,300>.

5.5.4. Registers and Function Unit

Figure GRAPHBLOCK shows the major functional components of the Sun graphics board. There are three data registers, destination, source, and mask, feeding into the function unit. These registers can be loaded from the host processor on a write cycle and from the frame buffer memory on a read cycle. There are three registers controlling update operation: function, width, and control. There are four sets of <x,y> registers for pixel addressing.

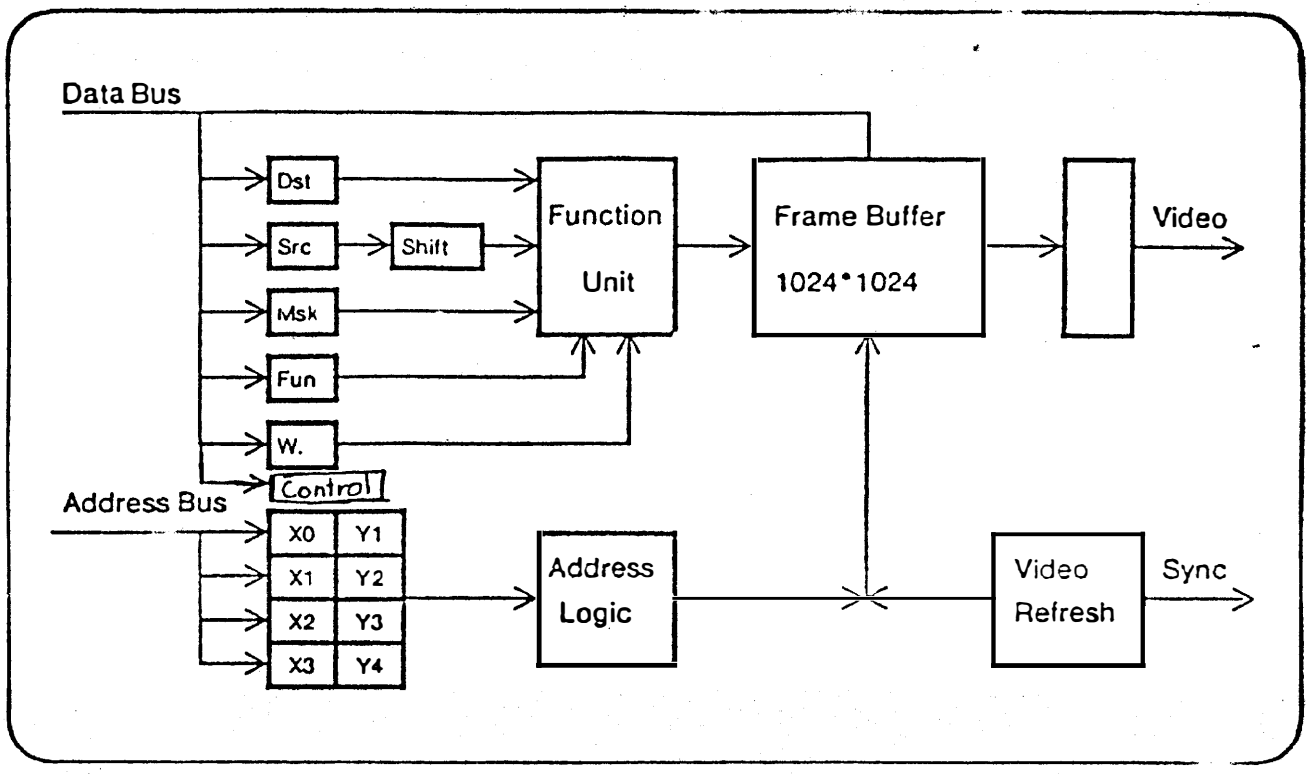


Figure GRAPHBLOCK. Sun Graphics Board Block Diagram.

5.5.4.1. Destination Register

The destination register holds the data that is being modified with a read-modify-write cycle on update operations in the frame buffer.

5.5.4.2. Source Register

The source register holds data to be combined with the destination data and the mask (pattern) data to compose new data for the frame buffer. The source register can be loaded from the frame buffer or from the processor. The data in the source register is bit-wise aligned with the bit-address of the destination in the frame buffer.

5.5.4.3. Mask Register

Similarly to the source register, the mask register holds data to be combined with the destination register and the source register to compose new data for the frame buffer. Again, the mask register can be loaded either from the frame buffer or from the Multibus. The difference between the mask register and the source register is that the mask register value is not bit-aligned with the <x> position of the destination in the frame buffer. Instead, the mask register is aligned to locations x such that (x mod 16 = 0), and is treated as a repeating pattern. The mask register is intended for background coloring and stipple-pattern generation where bit-alignment is undesirable.

5.5.4.4. Function Register

The function register specifies how the function unit combines destination, source, and mask data when data is written into the frame buffer memory. The eight-bit contents of the function register selects one of the 256 possible RasterOps for three boolean operands. See section "RasterOps" above for details.

5.5.4.5. Width Register

The actual width of an update operation is set via the width register from one to sixteen pixels.

5.5.4.6. Control Register

The control register controls video enable, interrupt enable, interrupt level, and graphics board LED as follows, where bits are numbered from D15 (most significant) to D0 (least significant):

D15..D13	Interrupt Level
D12..D11	Reserved
D10	LED Enable
D9	Video Enable
D8	Interrupt Enable

The Video Enable bit turns on video to the monitor; the screen appears blank when this bit is off. The Interrupt Enable bit enables interrupts on the level selected. When enabled, an interrupt is generated at the

beginning of every vertical retrace, allowing synchronization of display updates with display refresh. The Interrupt flag stays pending until reset in software by accessing the Interrupt Acknowledge location (see "Graphics Board Address Decoding" below for details). The LED Enable bit turns the graphics board LED off when set; the LED lights when this bit is zero.

The control register is cleared (set to zeros) on INIT to guarantee a blank screen, LED on, and disabled interrupts when the graphics board is powered up.

5.5.4.7. X-Y Registers

The host processor accesses graphical objects in the frame buffer via (x-y) register pairs. Four sets of (x-y) registers are provided that can be selected dynamically via the address bus. Only one (x) or (y) register is updated at one time; the others do not change.

5.5.5. Graphics Board Multibus Interface

The Sun graphics board uses both the data and the address lines of the Multibus to maximize the information that can be sent to the graphics board in one bus cycle. In a single cycle, the Sun processor can transfer a new 16-bit data item via the data bus and a new (x) or (y) address via the address bus. At the same time, the processor can select which of the four sets of (x,y) registers to use, whether to load a register from the data bus or from the frame buffer, whether to load the source or the mask register, and whether to execute a frame buffer update or not. Refer to the next section, "Graphics Board Address Decoding", for details on how this information is encoded on the Multibus.

On a write cycle, five things happen sequentially.

(1) One of the four sets of (x,y) "cursor" registers is selected.

(2) An x or y coordinate encoded in the address is loaded into the (x) or (y) register of the selected pair.

(3) Data from the Multibus is written into the selected register on the graphics board (source, mask, function, width, or control), or the data is ignored if no register is selected.

(4) The contents of the addressed (x,y) frame buffer location are read into the destination register.

(5) If and only if an update is requested, the data stored in the destination, source, and mask registers are combined according to the preselected function and new data is written back into the addressed frame buffer location.

On a read cycle, four things happen. The first two are the same as for the case of a write cycle.

(1) One of the four sets of (x,y) "cursor" registers is selected.

(2) An x or y coordinate encoded in the address is loaded into the (x) or (y) register of the selected pair.

(3) Data is read from the addressed (x,y) location in the frame buffer into the selected register on the graphics board.

(4) The data then stored in the source register is returned to the Multibus, correctly bit-aligned with the bus data lines.

The frame buffer is never updated on a read cycle and the update bit is ignored for read cycles.

A one-deep FIFO decouples the graphics board from the Sun processor. Processor requests are latched on the graphics board and are subsequently executed independently and in parallel with the processor. This makes the frame buffer a zero-access-time device as long as the request rate does not exceed one request per microsecond. Since normally streams of data are being transferred, the pipelining maximizes throughput.

On write cycles, the FIFO operation is invisible to the programmer. On read cycles, however, because of the pipelining, the data read back corresponds to the previous read request. Thus, to read a stream of data, one additional word needs to be read before valid data is obtained.

5.5.6. Graphics Board Address Decoding

The graphics board decodes 20 bits on the Multibus memory address lines, in the fields shown in Figure GXADDRESS. By encoding these operation bits in the address, repetitive operations like generalized RasterOps can be done very quickly. There is a patent pending on this design, which was meant to be used efficiently with MC68000 auto-incrementing addressing modes.

Up to eight graphics boards may share a single Multibus backplane, with the high 3 address bits (19 through 17) selecting the board. Each board occupies 128K bytes of Multibus address space. By accident this also happens to be how much physical memory there is on each graphics board.

The Update bit (bit 16) is on if the frame buffer is to be modified. Usually several operations are performed with this bit off, to set up the control registers and one of the coordinates. Then this bit is set to actually perform the desired modification of the frame buffer.

Bits 14 and 15 select the operation. If they are set to 0, then the data on the data bus is not used (although an X or Y address must be loaded in this cycle, as in all cycles). If they are set to 1, then one of the four auxiliary registers (Function, Width, Control, or Interrupt Clear) will be written with the data. If they are set to 2, the data bus is loaded into the "source" register. This is the normal case for

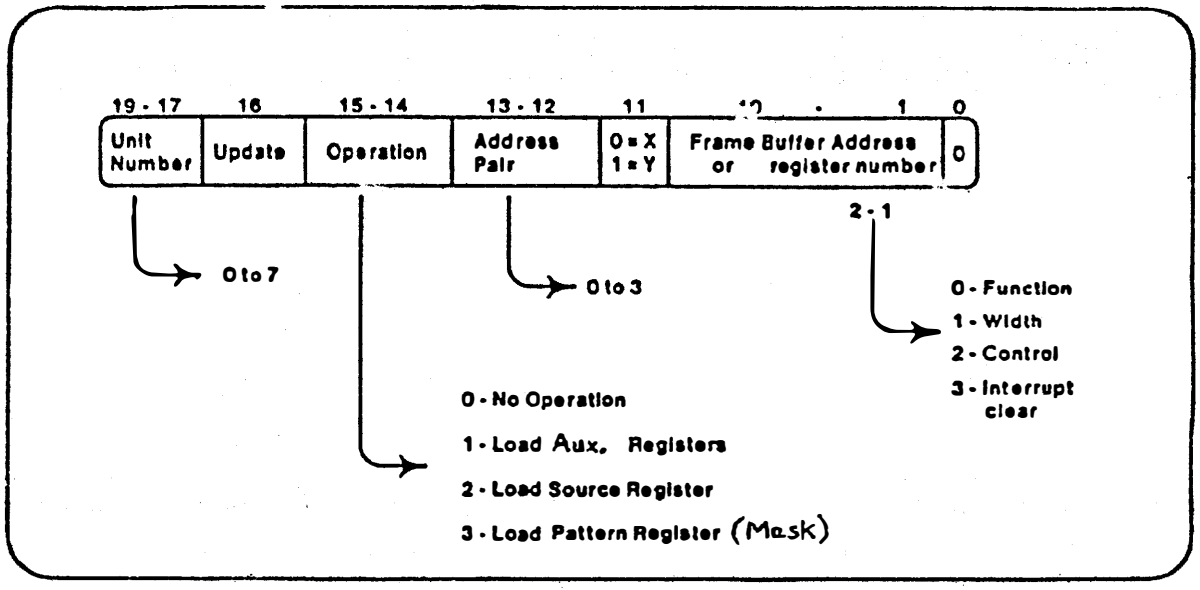


Figure GXADDRESS. Sun Graphics Board Address Decoding.

copy operations. If they are set to 3, the mask register will be loaded from the data bus. If the Update bit is also set in any of these cases, then the RasterOp will be performed and the frame buffer modified after the specified register is loaded, all in the same bus cycle.

When an Operation code of 1 is specified (auxiliary registers), the auxiliary register number is given in bits 1 and 2. A value of 0 loads the function register from the low-order eight bits of the data bus. The interpretation of the function register is explained above in section "RasterOps". A register number of 1 specifies the width register, which determines the width of the RasterOps. It is loaded from the low order 4 bits of the data bus, with 0 meaning 16, so its valid range is from 1 through 16. If it is less than 16, the high-order bits of the data in the source and pattern registers will be significant on RasterOps. An auxiliary register number of 2 loads the control register bits from the data bus. See section "Registers and Function Unit" above for details. Finally, register number 3 specifies Interrupt Clear. It must be accessed once after every video refresh interrupt to clear it, when the interrupt is enabled.

There are four pairs of ten-bit address registers (sometimes called "cursors"), selected by bits 12 and 13. Bit 11 selects either X or Y of the pair, and bits 1 through 10 of the address are loaded into the selected address register. Note that every read or write reference to the graphics board has to load one of these address registers, while it might or might not (depending on the Update and Operation code bits) modify the frame buffer.

The low order bit (bit 0) of the address must always be zero.

Appendix A contains a simple example (written in the programming language C) which displays an 8 by 8 "cursor" at a given screen position. The example also illustrates the use of some mnemonic definitions for the frame buffer commands. C language definitions for these names appear in Appendix B.

5.6. Terminal Emulation

The monitor provides routines which enable the Sun Workstation to emulate a standard ANSI terminal and a Tektronix 4014 terminal. The workstation utilized in this fashion is referred to in this section as the "Sun terminal". The Sun terminal is used by the monitor as a "Console" output device and is accessible to user programs through the ROM Vector Table routines finit(), fwritechar(), and fwrstr() (see "ROM Vector Table" above).

The Sun terminal displays 34 lines of 80 ASCII characters per line, with scrolling, (x,y) cursor addressability, and a number of other ANSI standard* control functions. In addition it provides all of the standard and many of the enhanced vector capabilities of the 4014**.

5.6.1. ANSI Terminal Emulation

The Sun terminal displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 and 0x7f inclusive are printing characters, which means that they have graphic renditions in the form of bit maps (see "Font Table", below). When a printing character is written to the Sun terminal and is not part of an escape sequence described below, it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see CTRL-J below), which causes the screen to scroll up by one or more lines, before moving the cursor to the first character position on the next line.

5.6.1.1. ANSI Control Sequence Syntax

The Sun terminal defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun terminal, it will not be displayed on the screen, but will effect some control function as described below, for example, move the cursor or set a display mode.

---

* ANSI Standard X3.64, "Additional Controls for Use with ASCII", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

** 4014 and 4014-1 Computer Display Terminal User's Manual, Tektronix, Inc., P.O. Box 500, Beaverton, Oregon 97077.

Some of the control sequences consist of a single character. The alternate notations

CTRL-X or ^X

for some character X, represent a control character which is typed on the keyboard by holding down the CTRL key and hitting X. On the Sun keyboard, the shift key has no effect on control characters; for example, CTRL-~ is equivalent to CTRL-^. See the end of the section titled "The Sun Keyboard" above.

Other ANSI control sequences are of the form

ESC [<params> <char>

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces. ESC represents the ASCII Escape character (ESC, CTRL-[, 0x1b). The next character is a left square bracket "[" (0x5b). The <params> are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons. <Char> represents a function character, which is different for each control sequence. Some examples of syntactically legal escape sequences are (again, ESC represent the single character Escape):

```
ESC[m
ESC[7m
ESC[33;54A
ESC[123;456;0;;3;B
```

Syntactically legal ANSI escape sequences which are not currently interpreted by the Sun terminal will be ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, then the remaining parameters are defaulted to 1, except as noted in the descriptions below. If more than the required number of parameters is supplied, then only the last n will be used, where n is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below). Consider, for example, the command character r which requires one parameter. ESC[;r and ESC[0r and ESC[r and ESC[23;15;32;1r are all equivalent to ESC[1r and provide a parameter value of 1. Note that ESC[;5r (interpreted as "ESC[5r") is NOT equivalent to ESC[5;r (interpreted as "ESC[1r"). In the syntax descriptions below, parameters are represented as "#" or "#1;#2".

5.6.1.2. ANSI Control Functions

The following paragraphs specify the ANSI control functions implemented by the Sun terminal. Each description gives:

the control sequence syntax

the hex equivalent of control characters where applicable

the ANSI standard control function name and two- or three-letter abbreviation where applicable

description of parameters required, if any

description of the control function

for functions which set a mode, the initial setting of the mode. The initial settings are restored on k1 or k2 reset, or by calling finit() (see "ROM Vector Table").

CTRL-G (0x7)
Bell

The Sun-1 Workstation is not equipped with an audible bell. It "rings the bell" by flashing the entire screen.

CTRL-H (0x8)
BACKSPACE

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

CTRL-I (0x9)
TAB

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

CTRL-J (0xA)
Line-feed

The cursor moves down one line, remaining at the same character position on the line.

If the cursor is already at the bottom line, the entire screen (including the cursor) is first scrolled up by s lines, where s is an internally saved value, initially 1, which can be changed by the ESC[r control sequence (see below). The top s lines scroll off the screen and are lost. S new blank lines scroll onto the bottom of the screen. After scrolling, the cursor moves down one line.

CTRL-L (0xC)
Form-feed

The cursor is postioned to the Home position (upper-left corner) and the entire screen is blanked.

CTRL-M (0xD)
RETURN

The cursor moves to the leftmost character position on the current line.

CTRL-[(0x1B)
ESC

This is the Escape character. It initiates a control sequence if followed immediately by a left bracket "[". Otherwise it is ignored.

ESC[#@
INSERT CHARACTER (ICH)

Takes one parameter, # (default 1). Makes room for # characters at the current cursor position by shifting to the right by # character positions the tail of the current line starting at the current cursor position inclusive. The # vacated positions at the cursor are filled with blanks; the rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A
CURSOR UP (CUU)

Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B
CURSOR DOWN (CUD)

Takes one parameter, # (default 1). Moves the cursor down # lines. if the cursor is fewer than # lines from the bottom of the screen, moves the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C
CURSOR FOWARD (CUF)

Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D
CURSOR BACKWARD (CUB)

Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E
CURSOR NEXT LINE (CNL)

Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, postions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f
HORIZONTAL AND VERTICAL POSITION (HVP)
or
ESC[#1;#2H
CURSOR POSITION (CUP)

These two escape sequences are equivalent. Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner).

ESC[J
ERASE IN DISPLAY (ED)

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K
ERASE IN LINE (EL)

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L
INSERT LINE (IL)

Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with blanks; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M
DELETE LINE (DL)

Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with blanks; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P
DELETE CHARACTER (DCH)

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC[#m
SELECT GRAPHIC RENDITION (SGR)

Takes one parameter, # (default 0). Note that, unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0	Normal rendition.
7	Negative (reverse) image.

Negative image will display characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and will select the negative image rendition.

ESC[p
Set screen mode to black-on-white.

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode blanks display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see "Select Graphic Rendition" above) will be white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC[q
Set screen mode to white-on-black.

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode blanks display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see "Select Graphic Rendition" above) will be black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC[#r
Set number of scroll lines.

Takes one parameter, # (default 1). Sets to # an internal register which determines how many lines the screen will scroll up when a line-feed function is performed with the cursor on the bottom line. See the description of the Line-feed (CTRL-J) control function above for details.

5.6.2. Vector-Drawing Control Functions

The Sun monitor's terminal emulator includes vector-drawing capabilities which are a subset of those implemented by the Tektronix 4014 terminal. The following modes are provided:

Alpha mode. ASCII characters are printed normally.

Graph mode. ASCII characters cause vectors to be drawn.

Point mode. ASCII characters cause points to be plotted.

Incremental mode. ASCII characters cause incremental "pen" motions.

5.6.2.1. Graph and Point Mode Address Format

In these modes a series of addresses are given. Each address causes the beam to move in a straight line from its current address to the given address. The first address of the series after entering graph mode is a dark vector (no vector is written), but all subsequent addresses cause vectors to be written. This is the only control there is over writing ("pen-up" versus "pen-down"). If the first address is preceded by ^G then the first vector is written rather than dark. In point mode, all vectors are dark, but the point at the given address is illuminated.

Addresses are transmitted either as full addresses or abbreviations thereof. A full address consists of two 10-bit coordinates Y,X transmitted in that order. Each coordinate is transmitted using two consecutive characters, with the high order 5 bits of the coordinate being extracted from the low order 5 bits of the first character and the the low order 5 bits of the coordinate being extracted from the low order 5 bits of the second character.

The high order 2 bits of each of the four characters transmitted for a full address must be respectively 01, 11, 01, 10. These denote respectively HI Y, LO Y, HI X, LO X. Whether 01 denotes HI Y or HI X is determined by whether LO X or LO Y respectively was transmitted most recently. On entering graph mode HI Y is assumed (as though LO X were transmitted most recently).

An address may be abbreviated merely by omitting characters, subject to the following two rules. LO X may never be omitted since it is the character that actually causes the vector to be drawn. HI X may occur only if immediately preceded by LO Y, since LO Y is used to enable the interpretation of 01 as HI X rather than as HI Y.

There is an extended address protocol to support two more low-order bits of precision. These bits are currently ignored by the Sun emulator since the Sun frame buffer does not have 4096x4096 resolution.

Two control characters are interpreted specially in these modes. CTRL-G (Bell) immediately following entry into graph mode causes the first vector to be drawn (pen down). NUL (0) is ignored in graph and point modes. All other control characters cause Alpha mode to be entered, (which is the normal mode in which the ANSI control functions are also available) and the character is interpreted in Alpha mode.

5.6.2.2. Incremental plotting mode

This mode is entered with ^^ (control-caret, 0x1E). Each subsequent character causes action according to the following table:

space	raises pen
P	lowers pen
A	move right
E	move right and up
D	move up
F	move left and up
B	move left
J	move left and down
H	move down
I	move right and down
NUL (^@)	ignored

All other characters cause exit from incremental plotting mode and are intrpreted as Alpha mode characters.

Note that each move occurs in increments of 1 in the 4014 "extended" 12-bit coordinate system. Since the Sun only uses 10-bit coordinates, no motion will be visible until at least four increments have occurred in the X or Y direction. In other words, the increments occur within 4x4 squares which are mapped to single pixels on the Sun frame buffer.

5.6.2.3. Control Sequences

The following table summarizes the control sequences which are relevant to vector drawing:

CTRL-] (0x1D) Enter graph mode, first vector dark, follwing vectors written.

CTRL-] CTRL-G (0x1D07) Enter graph mode, all following vectors written.

CTRL-\ (0x1C) Enter point mode

CTRL-^ (0x1E) Enter incremental plotting mode

5.6.3. Sun Terminal Font Table

The font table, which is accessible to user program via the ROM vector table, contains a bit map for each printable character. The font table is organized as an array of 96 elements; each element is an array of CHRSHORTS shortwords (CHRSHORTS is a constant currently defined as 20). Only the high-order CHRWIDTH bits of each (16-bit) shortword are used (CHRWIDTH is a constant currently defined as 12). An ASCII character c (32 <= c < 128) is displayed in a rectangular region 12 pixels wide by 22 pixels high, as follows:

(1) 32 is subtracted from c.

(2) The result is used as an index into FontTable, yielding an array of 20 shortwords.

(3) The topmost row of 12 pixels is displayed as white space.

(4) The next row of pixels from the top is displayed from the high-order 12 bits of the first shortword in the font table entry. The most significant bit appears leftmost on the screen.

(5) The next 19 rows of pixels are produced respectively from the next 19 shortwords in the font table entry.

(6) The bottom row of pixels is displayed as white space.

6. DIAGNOSTICS

Several levels of diagnostics exist for the Sun Workstation. These different sets of tests are used for the varied tasks of burning-in new systems at the factory, verifying system integrity at power-on in the field, and trouble-shooting machines which appear to be malfunctioning.

6.1. Factory Test Procedures

All SUN systems are subjected to several days of comprehensive testing and burn-in before they are shipped. These tests include testing at the component, board, and systems levels.

At the component level, "burned-in" chips are bought whenever possible; 64k RAMs are sent to an independent testing house for 72-hour burn-in at 125 degrees C. and electrical parameter testing.

At the next level of testing, boards go through a visual inspection and initial test. All boards must work between 20 to 70 degrees Celsius and 4.5 to 5.5 volts before they pass "Initial Test".

After IT, boards are put into a burn-in rack where they are subjected to power-cycling and high temperatures for a minimum of three days. The burned-in boards are then retested at the voltage margins.

In systems test, the machines go through a continuous power cycling. As the power comes on, each system verifies its internal operation and automatically boots a test program over the Ethernet. Each test program should then complete, and each machine then sends a verification of this fact to the main file server which records this fact. Each machine then selects at random a graphics demo program which it boots and runs until the next power cycle. System testing lasts a minimum of two days and continues until a machine is shipped.

6.2. Power-On Diagnostics

Every time a system is powered up, or on a monitor k2 reset, a 10-second set of ROM-based diagnostic tests is automatically performed. These diagnostics verify the memory management hardware, the timers and UARTs, on-board and Multibus memory, and the parity error detection circuitry. In the case of a failure, the ROM monitor will fail to boot. The "Sun Workstation Monitor" message will fail to appear on the screen and one of the LEDs on the processor card will blink at one second intervals to flag a hardware problem.

Should this happen, call your Sun Microsystems service organization for assistance. You also have the option of invoking a more comprehensive set of ROM-based diagnostics shipped with the machine.

6.3. Diagnostic PROMS

(NOTE: Do not attempt to repair the machine without authorization from your field service organization.)

To run the PROM-based diagnostics, replace the 8kx8 PROMs at processor board locations U101 and U103 with the diagnostic PROMs labelled "DIAG 0/0" and "DIAG 0/8" respectively. See "Installation of Diagnostic PROMs" below for detailed instructions. Connect an ASCII terminal set at 9600 baud to UART port A.

After the diagnostic PROMs have been installed, simply powering the machine on will run the diagnostics. An error-free run should display the following information:

```
UART & TIMER OK
NO BAD INTERRUPTS
TO
T1T2T3T4T5T6T7T8T9
T21T22T23T24T25T26T27T28T29
TESTING A:000000 TO A:040000
T41T42T43T44T45T46T47T48T49T50T51
T61
TESTING A:100000 TO A:120000
T81T82T83T84T85T86T87T88T89T90T91
TEST DONEx
```

The diagnostics will now cycle and run again.

Initially, the diagnostics test out the timer and UART circuitry. If there is an error at this point, the diagnostics will halt. If you cannot get any diagnostic output through the UART, contact your Sun Microsystems service organization for assistance.

Next, boot state is exited. In boot state, all interrupts are disabled, so while leaving boot state, it is possible that another board in the system is erroneously generating an interrupt when it should not be. If you fail to get the NO BAD INTERRUPTS message, remove the boards in the system (other than the processor board) one by one and re-run the diagnostics until the bad board has been identified.

The diagnostics next test out the context registers, the segment maps, the page maps, on-board memory, and Multibus memory. The diagnostics print out the test numbers as the tests are invoked, and any faulty chips are listed. Tests 1 to 9 diagnose the segment maps, tests 21 to 29 diagnose the page maps, tests 41 to 51 diagnose on-board memory, test 61 checks the 64K RAMs storing parity data, and tests 81 to 91 check Multibus memory. Multibus memory tests are not run if there is no Multibus memory in your system.

Please report all failures during the execution of these diagnostics to Sun Microsystems. In the case of a 64K RAM failure, a special diagnostic has been developed to isolate the fault down to the RAM chip level. Should T61 generate a failure, please refer to the section "Decoding 64K RAM Error Information" below for repair instructions.

6.4. Installation of Diagnostic PROMs

To install the Diagnostic PROMs:

1) Disconnect the power cord to the Sun-1.

2) Remove the six screws securing the tray on the sides of the SUN-1.

3) Pull out the tray.

4) Identify the processor board. This is the board with the ribbon cable which connects to the back panel keyboard connector (see figure BACKPANEL in section 1.2).

5) Remove any cables that are in the way, and extract the processor board.

6) With flat-head screwdriver or similar implement, gently remove the 28-pin packages at locations U101 and U103 which are labelled on the PC board.

7) Insert the diagnostic PROM labelled "0/0" at location U101.

8) Insert the diagnostic PROM labelled "0/8" at location U103.

9) Check that each PROM is inserted with pin 1 (the end with the small indentation) toward the nearest edge of the board, i.e. away from the Multibus connector.

10) Check that all pins on the PROMs have properly gone into their sockets.

11) Put the processor board back into the Multibus card cage.

12) Restore all cabling.

13) Connect an ASCII terminal to UART A. UART A (see Figure BACKPANEL in section 1.2) is the RS-232 connector attached to the section of the header on the processor board that is closest to the front of the system, immediately above the PROM slot labeled U102. The terminal should be set for 9600 baud.

14) Power the system ON, and the diagnostics will begin execution.

15) If the diagnostics show any error other than one in the format "BAD 64K A:000000 W:0000 R:0000", contact your Sun Microsystems service organization. To decode 64K RAM error information, see the next section.

6.5. Decoding 64K RAM Error Information

All 64K RAM errors display the address at which the error occurred, the data written to that location, and the data read from that location. All Addresses and data are displayed in hexadecimal. If the error is in

"on-board" memory (on the processor board or a Sun memory expansion board), a list of bad chip locations is also given. For example, if 0xAAAA was written to address 0x02E802 and 0x2AAB was read, the error displayed would be:

```
BAD 64K A:02E802 W:AAAA R:2AAB M0100 M0208
```

The bad chips are at locations M0100 and M0208 on the processor board and these can be replaced with 150ns 64K RAMs as supplied by Sun Microsystems for spares.

Note that bad chip locations are only shown for those boards in the on-board memory space. No chip locations are given for bad Multibus memory since the diagnostics cannot distinguish different types of Multibus memory boards.

6.5.1. Decoding On-Board RAM Locations

The memory locations given by the diagnostics are the same as those printed on the PC boards. However, these locations have been obscured by the sockets placed on the boards, so given a memory location, the thousands digit specifies the board number, the hundreds digit specifies the row in the memory array, and the ten and ones digits specify the column number in the memory array.

Clarifying the above statements, board 0 is the processor board; boards 1 and 2 are memory expansion boards 1 and 2 respectively; and location Mx100 is always oriented as the upper-left hand corner of the memory array (with the Multibus connector at the bottom of the card).

Take the memory location "M1108" as an example. The thousands digit specifies expansion board number 1. The 64K RAM is in the top row, and it is the eighth chip from the left-hand side of the board. Call your Sun Microsystems service organization if you have problems.

6.5.2. Decoding Chryslin 128K RAM Locations

On Chryslin 128K byte Multibus Memory Cards, the chip decoding is as follows:

The first 32K on the card correspond to the top row of chips (with the Multibus connector at the bottom of the card). The second, third, and fourth sets of 32K correspond, respectively, to the second, third, and fourth rows of chips on the card.

Data bits 7 through 0, where 0 is the least-significant, correspond to the eleventh through eighteenth columns. Data bits 15 through 8 correspond to the first through eighth columns. Columns nine and ten are parity chips.

The RAM chips used on Chryslin's 128K boards are 200ns 16K RAMs with +5 and -12 voltage levels. If these 16K RAMs are not readily available and you need temporary spares, you can either disable one section of 32K bytes on the board (UNIX, however, needs at least 128K bytes

of Multibus memory), or yank the parity chips from one section of 32K bytes and disable parity interrupts from the boards.

6.5.3. Decoding Chryslin 512K RAM Locations

The layout of the Chryslin 512K byte Multibus Memory Cards is nearly identical to that for their 128K byte Memory Cards. Each row of RAM chips, however, correspond to 128K bytes of the address space. See the preceding section for more information.

Appendix A. Frame Buffer Programming Example

This example presents a simple C language function called "DisplayCursor", which displays an 8 by 8 "cursor" at a given position. This program makes use of a number of mnemonic names, all of which begin with the letters GX, which are defined in the header file framebuf.h (see Appendix B). To perform an operation on the graphics board, address fields are combined together using logical OR ("|") and/or addition; the result is then referenced as a pointer, which puts the constructed address on the Multibus address lines, thereby effecting the desired operation on the graphics board. In this example, xPointer and yPointer are constructed and used in this fashion.

```
#include <framebuf.h>  /* See Appendix B. */
DisplayCursor( x, y )
short x, y;            /* screen coordinates of upper left corner */
{
    static short curscr[] = { 0x9200,
                              0x5400,  /* Left justified bit array */
                              0x3800,
                              0xFE00,
                              0x3800,
                              0x5400,
                              0x9200,
                              0x0000};
    register short
                *cursorPointer = cursor;
                *xPointer = (short *)(GXUnit0Base | GXselectX),
                *yPointer = (short *)(GXUnit0Base | GXupdate
                                      | GXsource | GXselectY);


/*  The following symbols are defined so as to load the width and
 *  function registers respectively:
 */
    GXwidth = 8;
    GXfunction = GXcopy;


/*  Since xPointer and yPointer are declared pointers to words (short *),
 *  the C compiler implicitly left-shifts x and y by one bit before
 *  adding them to the pointers (as required by the frame buffer):
 */
    xPointer += x;
    yPointer += y;


/*  The following statement sets the graphics board's x-register (of
 *  x-y pair 0) to x, which has been encoded into bits 1-10 of xPointer
 *  by the assignment statement above.  No other action is performed;
 *  the data value 0 is ignored.
```



```
 */
    *xPointer = 0;


/*  The following statement sets the graphics board's y-register (of
 *  x-y pair 0) to y;  it loads the source register from the Multibus
 *  data lines, which contain the first word of "cursor";
 *  and (since the GXupdate bit is set) it performs the specified
 *  RasterOp on the selected <x,y> position, i.e. copies source to
 *  destination.  The C compiler generates a single move instruction
 *  for this statement, with address postincrement on both operands.
 *  Since both yPointer and cursorPointer are pointers to words
 *  (short *), they are both incremented by two, which leaves them
 *  pointing in just the right place for the following instruction.
 *  (Recall that the frame buffer expects the y coordinate to be left-
 *  shifted one bit in the address.)
 */
    *yPointer++ = *cursorPointer++;


/*  Now continue copying the cursor to the screen. */
    *yPointer++ = *cursorPointer++;
    *yPointer++ = *cursorPointer++;
    *yPointer++ = *cursorPointer++;        /* (Each of these is one */
    *yPointer++ = *cursorPointer++;        /* 68000 move instruction) */
    *yPointer++ = *cursorPointer++;
    *yPointer++ = *cursorPointer++;
    *yPointer++ = *cursorPointer++;
}
```


Appendix B. C Language Constants for the Sun Graphics Board

This appendix contains the text of the C header file framebuf.h, which defines a number of useful mnemonic symbols for graphics board commands. These commands are encoded in bit fields within Multibus addresses which are decoded by the graphics board.

Several of these symbols are used in the program example in Appendix A.

```
/* framebuf.h
 * Copyright 1982 by Sun Microsystems, Inc.

 * The low order 11 bits consist of the X or Y address times 2.  The
 * lowest order bit is ignored, so word addressing works efficiently:
 */
# define GXselectX (0<<11)    /* the address is loaded into */
# define GXselectx (0<<11)                  /* an X register */

# define GXselectY (1<<11)    /* the address is loaded into */
# define GXselecty (1<<11)                   /* a Y register */

/*
 * There are four sets of X and Y register pairs, selected
 *   by the following bits:
 */
# define GXaddressSet0  (0<<12)
# define GXaddressSet1  (1<<12)
# define GXaddressSet2  (2<<12)
# define GXaddressSet3  (3<<12)
# define GXset0  (0<<12)
# define GXset1  (1<<12)
# define GXset2  (2<<12)
# define GXset3  (3<<12)

/*
 * The following bits indicate which registers are to be loaded:
 */
# define GXnone    (0<<14)
# define GXothers  (1<<14)
# define GXsource  (2<<14)
# define GXmask    (3<<14)
# define GXpat     (3<<14)


# define GXupdate (1<<16)       /* actually update the frame buffer */


/*
 * These registers can appear on the left of an assignment statement.
 *  Note they clobber X register 3:
```



```
 */
# define GXfunction     *(short *)(GXBase + GXset3 + GXothers + (0<<1))
# define GXwidth        *(short *)(GXBase + GXset3 + GXothers + (1<<1))
# define GXcontrol      *(short *)(GXBase + GXset3 + GXothers + (2<<1))
# define GXintClear     *(short *)(GXBase + GXset3 + GXothers + (3<<1))

# define GXsetMask      *(short *)(GXBase + GXset3 + GXmask)
# define GXsetSource    *(short *)(GXBase + GXset3 + GXsource)
# define GXpattern      *(short *)(GXBase + GXset3 + GXpat)

/*
 * The following bits are written into the GX control register.
 * It is reset to zero on hardware reset and power-up.
 * The high order three bits determine the Interrupt level (0-7):
 */
# define GXintEnable    (1<<8)
# define GXvideoEnable  (1<<9)
# define GXintLevel0    (0<<13)
# define GXintLevel1    (1<<13)
# define GXintLevel2    (2<<13)
# define GXintLevel3    (3<<13)
# define GXintLevel4    (4<<13)
# define GXintLevel5    (5<<13)
# define GXintLevel6    (6<<13)
# define GXintLevel7    (7<<13)

/*
 * The following are "function" encodings loaded into
 *         the function register:
 */
# define GXnoop                 0xAAAA
# define GXinvert               0x5555
# define GXcopy                 0xCCCC
# define GXcopyInverted         0x3333
# define GXclear                0x0000
# define GXset                  0xFFFF
# define GXpaint                0xEEEE
# define GXpaintInverted        0x2222
# define GXxor                  0x6666
# define GXor   GXpaint

/*
 * These may appear in statement contexts to just set the X and Y
 *  registers of set number zero to the given values:
 */
# define GXsetX(X)   *(short *)(GXBase + GXselectX + ((X)<<1)) = 1
# define GXsetY(Y)   *(short *)(GXBase + GXselectY + ((Y)<<1)) = 1
```


Appendix C. ROM Vector Table Header File

This appendix contains the C-language header file which defines the Sun Monitor ROM Vector Table.

```
/*
 * sunromvec.h
 * Copyright 1982 by Sun Microsystems, Inc.

 * Sun Monitor ROM Vector
 *
 * This vector table is the only knowledge the outside world has of
 * this prom.  It is referenced by hardware (Reset SSP, PC), things
 * in the 2nd prom, and programs & operating systems that run under
 * the monitor.  Once located, no entry can be re-located unless you
 * change the world that needs it.
 *
 * The entries start at location 0x200000 [very first thing in the
 * Prom].
 *
 * The easiest way to reference elements of this vector is to say:
 *
 *      *RomVecPtr->xxx
 *
 * as in:
 *
 *      (*RomVecPtr->v_putchar)(c);
 *
 * This is pretty cheap as it only generates   movl 0x2000xx,a0  jsr a0@
 * That's 2 bytes longer, and 4 cycles longer, than a simple subroutine
 * call.
 *
 * If you don't want your programs to depend on the absolute location of
 * the EPROMs in current Sun processor boards, you can reference this
 * vector via a pointer in location 4 instead.  Just replace "RomVecPtr"
 * in the expression with "RomVecAddr".  This indirects thru location 4,
 * so it takes another 4 bytes of move instructions (16 cycles) per
 * reference ( movl 0x4,a0; movl a0@(xx),a0; jsr a0@).
 */

struct sunromvec {

        char *v_initsp;      /* Initial SSP for hardware RESET */
        int (*v_startmon)(); /* Initial PC  for hardware RESET */
/* Monitor and hardware revision and identification */
        char *v_SunRev;      /* Revision level of monitor & hardware */
        long *v_SerialNum;   /* Serial number of this Sun */
        long *v_MemorySize;  /* Physical onboard memory size */
/* Monitor-managed single-character input and output */
        unsigned char (*v_getchar)();/* Get char from cur input source */
        int (*v_putchar)();  /* Put char to current output sink */
        int (*v_mayget)();   /*Maybe get char from current input source*/
```

```
        int (*v_mayput)();   /* Maybe put char to current output sink */
        unsigned char *v_EchoOn;/* Should getchar echo its input? */
        unsigned char *v_InSource;/* Input source selector */
        unsigned char *v_OutSink;       /* Output sink selector */
/* Monitor-managed keyboard input (scanned by monitor refresh routine) */
        int (*v_getkey)();   /* Get next translated key if one exists */
        int (*v_InitGetkey)(); /* Initialize before first getkey */
        unsigned int *v_translation;/* Keyboard translation selector */
        unsigned char *v_KeybId;        /* Up/down keyboard ID byte */
        keyboard **v_CurKeyboard;/* Address of current keyboard defn
                                                            struct */
        keymap * (*v_SetTable)();/* Address of current shift->table
                                                            mapper */
        keybuftype *v_Keybuf;   /* Up/down keycode buffer */
/* Monitor-managed framebuffer output and terminal emulation */
        int (*v_finit)();       /* Framebuffer initialization routine */
        int (*v_fwritechar)(); /* Write a character to FB "terminal" */
        long *v_FBAddr;    /* Address of frame buffer we are using */
        char **v_FontTable;/* Address of font table for FB "terminal" */
        int (*v_fwrstr)();      /* Write a string to FB -- faster */
/* Monitor-managed mouse input (scanned by monitor refresh routine) */
        int *v_MouseBuf;        /* Placeholder for Mouse input buffer */
/* Monitor-managed line input and parsing */
        unsigned char *v_linbuf;     /* The line input buffer */
        unsigned char **v_lineptr;   /* Current pointer into linbuf */
        int *v_linesize;    /* Total length of line in linbuf */
        int (*v_getline)(); /* Fill linbuf from current input source */
        unsigned char (*v_getone)();    /* Get next char from linbuf */
        unsigned char (*v_peekchar)();/* Peek at next char
                                        without reading it */
        int (*v_gethexbyte)();/* Get next 2 chars and xlate to binary */
        int (*v_getnum)();   /* Get next numerics and xlate to binary */
/* Monitor-managed phrase output to current output sink */
        int (*v_message)(); /* Print a null-terminated string */
        int (*v_printhex)(); /* Print N digits of a longword in hex */
/* Memory management routines which must run outside mappable
                                                    address space */
        int (*v_SetSeg)();   /* Set segment map entry X in context C */
        short (*v_GetSeg)(); /* Get segment map entry X from context C */
/* Refresh related information */
        int (*v_KeyFrsh)();/* Address that oughta be in level 7 vector */
        int (*AbortEnt)(); /* Monitor entry point from keyboard abort */
        int *v_RefrCnt;    /* Refresh routines's millisecond count */
/* Assorted useful things added after the first bunch */
        int (*v_bedecode)(); /* Bus error decoding routine */
        int (**v_keyentry)();/* Called when table has unknown entry */
/* Monitor internal interface information */
        GlobDes *v_GlobPtr;  /* Address of monitor global variables */
        int (*v_diagret)();  /* Re-entry point for diagnostics */
};

#define  RomVecPtr ((struct sunromvec *)0x200000)
#define  RomVecAddr (*(struct sunromvec **)0x4)
```


Appendix D. Sun Keyboard Translation Table Definitions

This appendix contains the C-language definitions for the Sun keyboard.

```
/* keyboard.h
 * Copyright 1982 by Sun Microsystems, Inc.

 * Header file for Sun Microsystems parallel keyboard routines
 *
 * The keyboard is a standard Micro Switch, or otherwise, keyboard,
 * with an 8048/8748 on the board.  This has been modified to produce
 * up/down keycodes on 8 parallel output lines.  Each keycode is held
 * stable on these lines for a minimum of 2.5 ms in order that the
 * main processor can read it during its refresh routine, which executes
 * every 2 ms or so.
 *
 * When no keys are depressed, the keyboard transmits a keycode of
 * 7F hex, to indicate that.  Thus, when the last key is released,
 * a keycode for its release is sent, then a 7F.
 *
 * Eventually the 8048 will, when idle, alternate its 7F's with
 * keyboard id codes, identifying the key layout or other factors
 * which host programs might want to automatically determine.
 * The support for this feature is commented out with "#ifdef KBDID"
 * and is currently untested.
 */

#ifndef GETKEYPORT

/*
 *      How to get the current keycode being presented by the 8048
 *
 *      GETKEYPORT reads the raw port; GETKEY deglitches it.
 */
#define GETKEYPORT      ( *(unsigned char*) 0xE00000 )
#define GETKEY(key) \
        while (GETKEYPORT != ( (key) = GETKEYPORT ) );

/*
 *      Various special characters that might show up on the port
 */
#define NOTPRESENT      0xFF    /* Keyboard is not plugged in */
#define IDLEKEY         0x7F    /* Keyboard is idle; i.e. no keys down */
/* See comments above re keyboard id's which might follow IDLEKEYs. */
#define PRESSED         0x00    /* 0x80 bit off means key was pressed */
#define RELEASED        0x80    /* 0x80 bit on  means key was released */

#define ABORTKEY1       0x01    /* First key of abort seq - ERASE EOF */
#define ABORTKEY2         77    /* 2nd key of abort seq - "A" */
/* The above is horribly Micro Switch 103SD dependent. */
```



```
/*
 *      Keyboard ID codes...transmitted by the various keyboards
 *      after the IDLEKEY code.  See top of file for more details.
 */
#define KB_UNKNOWN      0x00    /* Unknown keyboard */
#define KB_MS_103SD32   0x00    /* Micro Switch 103SD32-2 */
#define KB_MS_VT100     0x01    /* Micro Switch VT100 compatible
                                                        SD-02499 */

/*
 *      These are the states that the keyboard scanner can be in.
 *
 *      It starts out in AWAITIDLE state.
 */
#define AWAITIDLE       0       /*Waiting til all keys up and ID sent*/
#define NORMAL          1       /*The usual (ho, hum)*/
#define ABORT1          2       /*Last key down was KEYABORT1*/
#define IDLE1           3       /*Last keycode was IDLEKEY -- ID follows*
#define IDLE2           4       /*Last keycode was id -- if IDLEKEY
                                    follows, don't put it in the buffer. *

/*
 *      Translation options for getkey()
 *
 *      These are how you can have your input translated.
 */
#define TR_NONE         0
#define TR_ASCII        1

/*
 *      These bits can appear in the result of TR_NONE getkey()s.
 */
#define STATEOF(key)    ( key & 0x80 )  /* 0=key down, 0x80=key up */
#define KEYOF(key)      ( key & 0x7F )  /* The key number that moved */
#define NOKEY           ( -1 )          /* The argument was 0, and no
                                                   key was depressed. */

/*
 *      These bits can appear in the result of TR_ASCII getkey()s.
 */
/* NOKEY can appear if the getkey()'s arg was 0 (no waiting) */
#define METABIT         0       /* Meta key depressed with key */
#define   METAMASK        0x000080
/* other "bucky" bits can be defined at will.  See "BUCKYBITS" below. */

/*
 *      This defines the bit positions used within "shiftmask" to
 *      indicate the "pressed" (1) or "released" (0) state of shift keys.
 *      Both the bit numbers, and the aggregate masks, are defined.
 *
 *      The "UPMASK" is a minor kludge.  Since whether the key is going
 *      up or down determines the translation table (just as the shift
 *      keys' positions do), we OR it with "shiftmask" to get "tempmask",
 *      which is the mask which is actually used to determine the
```



```
 *      translation table to use.  Don't reassign 0x0080 for anything
 *      else, or we'll have to shift and such to squeeze in UPMASK,
 *      since it comes in from the hardware as 0x80.
 */
#define CAPSLOCK        0       /* Caps Lock key */
#define   CAPSMASK        0x0001
#define SHIFTLOCK       1       /* Shift Lock key */
#define LEFTSHIFT       2       /* Left-hand shift key */
#define RIGHTSHIFT      3       /* Right-hand shift key */
#define   SHIFTMASK       0x000E
#define LEFTCTRL        4       /* Left-hand control key */
#define RIGHTCTRL       5       /* Right-hand control key */
#define   CTRLMASK        0x0030
/* unused...            0x0040 */
#define   UPMASK          0x0080

/*
 *      This defines the format of translation tables.
 *
 *      A translation table is 128 bytes of "entries", which are bytes
 *      (unsigned chars).  The top 4 bits of each entry are decoded by
 *      a case statement in getkey.c.  If the entry is less than 0x80,
 *      it is sent out as an ASCII character (possibly with bucky bits
 *      OR-ed in).  "Special" entries are 0x80 or greater, and invoke
 *      more complicated actions.
 */
typedef unsigned char   keymap [128]; /* maps keycodes to actions */

typedef struct {
        keymap          *Normal, *Shifted, *Caps, *Control, *Up;
        int             IdleShifts;
        } keyboard;             /* All keymaps of a keyboard, plus. */

#define SHIFTKEYS  0x80 /* thru 0x8F.  This key helps to determine the
                           translation table used.  The bit
                           position of its bit in "shiftmask"
                           is added to the entry, eg
                           SHIFTKEYS+LEFTCTRL.  When this entry is
                           invoked, the bit in "shiftmask" is
                           toggled.  Depending which tables you put
                           it in, this works well for hold-down
                           keys or press-on, press-off keys.  */
#define BUCKYBITS  0x90 /* thru 0x9F.  This key determines the state of
                           one of the "bucky" bits above the
                           returned ASCII character.  This is
                           basically a way to pass mode-key-up/down
                           information back to the caller with each
                           "real" key depressed.  The concept, and
                           name "bucky" (derivation unknown) comes
                           from the MIT/SAIL "TV" system...they had
                           TOP, META, CTRL, and a few other bucky
                           bits.  The bit position of its bit in
                           "buckybits", minus 7, is added to the
                           entry; eg bit 0x00000400 is BUCKYBITS+3.
```



```
                                The "-7" prevents us from messing up the
                                ASCII char, and gives us 16 useful bucky
                                bits.  When this entry is invoked,
                                the designated bit in "buckybits" is
                                toggled.  Depending which tables you put
                                it in, this works well for hold-down
                                keys or press-on, press-off keys.  */
#define FUNNY           0xA0 /* thru 0xAF.  This key does one of 16 funny
                                things based on the low 4 bits: */
#define NOP             0xA0 /* This key does nothing. */
#define OOPS            0xA1 /* This key exists but is undefined. */
#define HOLE            0xA2 /* This key does not exist on the keyboard.
                                Its position code should never be
                                generated.  This indicates a software/
                                hardware mismatch, or bugs. */
/* combinations 0xA3 - 0xAF are reserved for more funnies. */

#define STRING          0xB0 /* thru 0xBF.  The low-order 4 bits index
                                a table referenced via gp->KStrTab and
                                select a string to be returned, char
                                by char.  Each entry in KStrTab is a
                                pointer to a character vector.  The vector
                                contains a 1-byte length, followed by
                                data. */
/* Definitions for the individual string numbers: */
#define HOME                    0x00
#define UP                      0x01
#define DOWN                    0x02
#define LEFT                    0x03
#define RIGHT                   0x04
/* string numbers 5 thru F are reserved.  Users making custom entries,
        take them from F downward to postpone confusion. */

/* combinations 0xC0-0xFF reserved for more argument-taking entries */

#endif GETKEYPORT

/*
 *      End of keyboard.h
 */
```



```
/*****************************************************************************
 *
 * keytables.c
 * Copyright 1982 by Sun Microsystems, Inc.

 *
 * This module contains the translation tables for the up-down encoded
 * Sun keyboard.
 *
 * A fuller explanation of the entries is in keyboard.h.
 */

#include "keyboard.h"

/* handy way to define control characters in the tables */
#define c(char) (char-0x40)
#define ESC 0x1B

/* Unshifted keyboard table for Micro Switch 103SD32-2 */

keymap KTMSlc[1] = {
/*  0 */ HOLE,    OOPS,    OOPS,    OOPS,    HOLE,    OOPS,    OOPS,    OOPS,
/*  8 */ OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,
/* 16 */ OOPS,    OOPS,    OOPS,    c('['), HOLE,    OOPS,    '+',     '-',
/* 24 */ HOLE,    OOPS,    '\f',    OOPS,    HOLE,    SHIFTKEYS+CAPSLOCK,
                                                      '1',     '2',
/* 32 */ '3',     '4',     '5',     '6',     '7',     '8',     '9',     '0',
/* 40 */ '-',     '~',     '`',     '\b',    HOLE,    '7',     '8',     '9',
/* 48 */ HOLE,    OOPS,    STRING+UP,
                                    OOPS,    HOLE,    '\t',    'q',     'w',
/* 56 */ 'e',     'r',     't',     'y',     'u',     'i',     'o',     'p',
/* 64 */ '{',     '}',     '_',     HOLE,    '4',     '5',     '6',     HOLE,
/* 72 */ STRING+LEFT,
                    STRING+HOME,
                             STRING+RIGHT,
                                      HOLE,    SHIFTKEYS+SHIFTLOCK,
                                                        'a',     's',     'd',
/* 80 */ 'f',     'g',     'h',     'j',     'k',     'l',     ';',     ':',
/* 88 */ '|',     '\r',    HOLE,    '1',     '2',     '3',     HOLE,    OOPS,
/* 96 */ STRING+DOWN,
                    OOPS,    HOLE,    HOLE,    SHIFTKEYS+LEFTSHIFT,
                                                        'z',     'x',     'c',
/*104 */ 'v',     'b',     'n',     'm',     ',',     '.',     '/',     SHIFTKEY
                                                                        +RIGHTSH
/*112 */ NOP,     0x7F,    '0',     NOP,     '.',     HOLE,    HOLE,    HOLE,
/*120 */ HOLE,    HOLE,    SHIFTKEYS+LEFTCTRL,
                                      ' ',     SHIFTKEYS+RIGHTCTRL,
                                                        HOLE,    HOLE,    HOLE,
};

/* Shifted keyboard keymap for Micro Switch 103SD32-2 */

keymap KTMSuc [1] = {
/*  0 */ HOLE,    OOPS,    OOPS,    OOPS,    HOLE,    OOPS,    OOPS,    OOPS,
```

```
/*  8 */ OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,
/* 16 */ OOPS,    OOPS,    OOPS,    c('['), HOLE,    OOPS,    '+',     '-',
/* 24 */ HOLE,    OOPS,    '\f',    OOPS,    HOLE,    SHIFTKEYS+CAPSLOCK,
                                                      '!',     '"',
/* 32 */ '#',     '$',     '%',     '&',     '\'',    '(',     ')',     '0',
/* 40 */ '=',     '^',     '@',     '\b',    HOLE,    '7',     '8',     '9',
/* 48 */ HOLE,    OOPS,    STRING+UP,
                                    OOPS,    HOLE,    '\t',    'Q',     'W',
/* 56 */ 'E',     'R',     'T',     'Y',     'U',     'I',     'O',     'P',
/* 64 */ '[',     ']',     '_',     HOLE,    '4',     '5',     '6',     HOLE,
/* 72 */ STRING+LEFT,
                  STRING+HOME,
                           STRING+RIGHT,
                                    HOLE,    SHIFTKEYS+SHIFTLOCK,
                                                      'A',     'S',     'D',
/* 80 */ 'F',     'G',     'H',     'J',     'K',     'L',     '+',     '*',
/* 88 */ '\\',    '\r',    HOLE,    '1',     '2',     '3',     HOLE,    OOPS,
/* 96 */ STRING+DOWN,
                  OOPS,    HOLE,    HOLE,    SHIFTKEYS+LEFTSHIFT,
                                                      'Z',     'X',     'C',
/*104 */ 'V',     'B',     'N',     'M',     '<',     '>',     '?',     SHIFTKEY
                                                                        +RIGHTSH
/*112 */ NOP,     0x7F,    '0',     NOP,     '.',     HOLE,    HOLE,    HOLE,
/*120 */ HOLE,    HOLE,    SHIFTKEYS+LEFTCTRL,
                                    ' ',     SHIFTKEYS+RIGHTCTRL,
                                                      HOLE,    HOLE,    HOLE,
};


/* Caps Locked keyboard keymap for Micro Switch 103SD32-2 */

keymap KTMScl [1] = {
/*  0 */ HOLE,    OOPS,    OOPS,    OOPS,    HOLE,    OOPS,    OOPS,    OOPS,
/*  8 */ OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,    OOPS,
/* 16 */ OOPS,    OOPS,    OOPS,    c('['), HOLE,    OOPS,    '+',     '-',
/* 24 */ HOLE,    OOPS,    '\f',    OOPS,    HOLE,    SHIFTKEYS+CAPSLOCK,
                                                      '1',     '2',
/* 32 */ '3',     '4',     '5',     '6',     '7',     '8',     '9',     '0',
/* 40 */ '-',     '~',     '`',     '\b',    HOLE,    '7',     '8',     '9',
/* 48 */ HOLE,    OOPS,    STRING+UP,
                                    OOPS,    HOLE,    '\t',    'Q',     'W',
/* 56 */ 'E',     'R',     'T',     'Y',     'U',     'I',     'O',     'P',
/* 64 */ '{',     '}',     '_',     HOLE,    '4',     '5',     '6',     HOLE,
/* 72 */ STRING+LEFT,
                  STRING+HOME,
                           STRING+RIGHT,
                                    HOLE,    SHIFTKEYS+SHIFTLOCK,
                                                      'A',     'S',     'D',
/* 80 */ 'F',     'G',     'H',     'J',     'K',     'L',     ';',     ':',
/* 88 */ '|',     '\r',    HOLE,    '1',     '2',     '3',     HOLE,    OOPS,
/* 96 */ STRING+DOWN,
                  OOPS,    HOLE,    HOLE,    SHIFTKEYS+LEFTSHIFT,
                                                      'Z',     'X',     'C',
/*104 */ 'V',     'B',     'N',     'M',     ',',     '.',     '/',     SHIFTKEY
```

```
                                                                    +RIGHTSH
/*112 */ NOP,    0x7F,   '0',    NOP,    '.',    HOLE,   HOLE,   HOLE,
/*120 */ HOLE,   HOLE,   SHIFTKEYS+LEFTCTRL,
                                ' ',    SHIFTKEYS+RIGHTCTRL,
                                        HOLE,   HOLE,   HOLE,
};

/* Controlled keyboard keymap for Micro Switch 103SD32-2 */

keymap KTMSct [1] = {
/*  0 */ HOLE,   OOPS,   OOPS,   OOPS,   HOLE,   OOPS,   OOPS,   OOPS,
/*  8 */ OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,
/* 16 */ OOPS,   OOPS,   OOPS,   c('['), HOLE,   OOPS,   OOPS,   OOPS,
/* 24 */ HOLE,   OOPS,   '\f',   OOPS,   HOLE,   SHIFTKEYS+CAPSLOCK,
                                                        OOPS,   OOPS,
/* 32 */ OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,   OOPS,
/* 40 */ OOPS,   c('^'), c('@'), '\b',   HOLE,   OOPS,   OOPS,   OOPS,
/* 48 */ HOLE,   OOPS,   STRING+UP,
                                OOPS,   HOLE,   '\t',   c('Q'), c('W'),
/* 56 */ c('E'), c('R'), c('T'), c('Y'), c('U'), c('I'), c('O'), c('P'),
/* 64 */ c('['), c(']'), c('_'), HOLE,   OOPS,   OOPS,   OOPS,   HOLE,
/* 72 */ STRING+LEFT,
                STRING+HOME,
                        STRING+RIGHT,
                                HOLE,   SHIFTKEYS+SHIFTLOCK,
                                                c('A'), c('S'), c('D'),
/* 80 */ c('F'), c('G'), c('H'), c('J'), c('K'), c('L'), OOPS,   OOPS,
/* 88 */ c('\\'),
                '\r',   HOLE,   OOPS,   OOPS,   OOPS,   HOLE,   OOPS,
/* 96 */ STRING+DOWN,
                OOPS,   HOLE,   HOLE,   SHIFTKEYS+LEFTSHIFT,
                                                c('Z'), c('X'), c('C'),
/*104 */ c('V'), c('B'), c('N'), c('M'), OOPS,   OOPS,   OOPS,   SHIFTKEY
                                                                    +RIGHTSH
/*112 */ NOP,    0x7F,   OOPS,   NOP,    OOPS,   HOLE,   HOLE,   HOLE,
/*120 */ HOLE,   HOLE,   SHIFTKEYS+LEFTCTRL,
                                '\0',   SHIFTKEYS+RIGHTCTRL,
                                        HOLE,   HOLE,   HOLE,
};


/* "Key Up" keyboard keymap for Micro Switch 103SD32-2 */

keymap KTMSup [1] = {
/*  0 */ HOLE,   NOP,    NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,
/*  8 */ NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,
/* 16 */ NOP,    NOP,    NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,
/* 24 */ HOLE,   NOP,    NOP,    NOP,    HOLE,   SHIFTKEYS+CAPSLOCK,
                                                        NOP,    NOP,
/* 32 */ NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,
/* 40 */ NOP,    NOP,    NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,
/* 48 */ HOLE,   NOP,    NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,
/* 56 */ NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,
/* 64 */ NOP,    NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,    HOLE,
```



```
/* 72 */ NOP,    NOP,    NOP,    HOLE,   SHIFTKEYS+SHIFTLOCK,
                                                NOP,    NOP,    NOP,
/* 80 */ NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,
/* 88 */ NOP,    NOP,    HOLE,   NOP,    NOP,    NOP,    HOLE,   NOP,
/* 96 */ NOP,    NOP,    HOLE,   HOLE,   SHIFTKEYS+LEFTSHIFT,
                                                NOP,    NOP,    NOP,
/*104 */ NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    NOP,    SHIFTKEY
                                                                +RIGHTSH
/*112 */ NOP,    NOP,    NOP,    NOP,    NOP,    HOLE,   HOLE,   HOLE,
/*120 */ HOLE,   HOLE,   SHIFTKEYS+LEFTCTRL,
                        NOP,    SHIFTKEYS+RIGHTCTRL,
                                                HOLE,   HOLE,   HOLE,
};


/* Index to keymaps for Micro Switch 103SD32-2 */
keyboard KIMS [1] = { KTMSlc, KTMSuc, KTMScl, KTMSct, KTMSup, 0x0000 };

/*****************************************************************************/
/*   Index table for the whole shebang                                       */
/*****************************************************************************/
keyboard *KeyTables[] = { KIMS };

/*
        Keyboard String Table

        This defines the strings sent by various keys (as selected in
        the tables above).

        The first byte of each string is its length, the rest is data.
*/

char StrHome[] = "\003\033[H";          /* home */
char StrUp[] = "\003\033[A";            /* up*/
char StrDown[] = "\003\033[B";          /* down */
char StrLeft[] = "\003\033[D";          /* lef */
char StrRight[] = "\003\033[C";         /* Right */
char StrEmpty[] = "";                   /* unused string */

char *KeyStringTab[16] = {   /* foo */
        StrHome, StrUp, StrDown, StrLeft,
        StrRight, StrEmpty, StrEmpty, StrEmpty,
        StrEmpty, StrEmpty, StrEmpty, StrEmpty,
        StrEmpty, StrEmpty, StrEmpty, StrEmpty,
}
```


sun microsystems, inc.

9 September 1982

ERRATA for SUN-1 System Reference Manual, Version 1.0 Draft of 27 July, 1982

Please note the following errors in the manual:

Page 2, Figure BACKPANEL. The SMD connectors have been repositioned. A new figure will be included in the next revision.

Page 14, Card cage configuration. The diagram and description of the 7-slot card cage is not correct. It does correctly descirbe the 6-slot card cage, if slot 7 is deleted and the graphics card is placed in slot 6. A new figure will be included in the next revision.

Page 16, last paragraph. "16-pin" should read 50-pin".

Page 18, last paragraph. The instructions are incorrect. The knob for unlocking the heads on the Fujitsu is located underneath the unit. It is accessed through a hole near the center of the bottom of the enclosure.

Page 8, Memory Expansion. replace sentences. "insert the 74S138 chip at location U1006." with select dipswitch U506 to position 1.

2310 WALSH AVENUE, SANTA CLARA, CA 95051 (408) 748-9900